FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Reproduzierbarkeit von Ergebnissen im Bereich Maschinelles Lernen

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Computational Intelligence

eingereicht von

BCS Naumovska Martina
Matrikelnummer 1428441

an der Fakultät für Informatik
der Technischen Universität Wien
Betreuung: Ao.univ.Prof. Dr. Rauber Andreas

Wien, 27. Dezember 2016

Naumovska Martina	Rauber Andreas

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Reproducibility of machine learning results

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Computational Intelligence

by

BCS Naumovska Martina
Registration Number 1428441

to the Faculty of Informatics
at the TU Wien
Advisor: Ao.univ.Prof. Dr. Rauber Andreas

Vienna, 27th December, 2016

Naumovska Martina	Rauber Andreas

Erklärung zur Verfassung der Arbeit

BCS Naumovska Martina
Medwegweg 3/415, 1011 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 27. Dezember 2016

Naumovska Martina

Danksagung

Ich danke und bedanke mich bei Ao.univ.Prof. Dr. Andreas Rauber für die Gelegenheit, Anleitung und Unterstützung bei der Arbeit an dieser Arbeit.

Ich danke Dipl.-Ing. Ing. Thomas Lidy für seine Hilfe erklärte mir die Feature Extraktoren und psycho-akustische Transformationen für Musik Genre Klassifizierung Prozess. Seine Zeit, Hingabe und Hilfe ist wirklich geschätzt.

Ich bin auch Guillermo Herrero Igeño dankbar, weil ich den ersten Entwurf gelesen und mich beim Schreiben begleitet habe.

Ich danke meinen Freunden Damjan Gerovski, Magdalena Piponska, Ljuben Necevki Iva Shokoska i Iva Nikolic für ihre bedingungslose Liebe, Unterstützung und Verständnis während dieses Prozesses.

Schließlich möchte ich meinen Eltern für ihre bedingungslose Liebe, den Glauben, die Unterstützung und die Annahme an mich danken. Ohne sie wäre diese Anerkennung nicht möglich. Meine Errungenschaften sind auch immer ihre Errungenschaften. Wenig aber nicht zuletzt möchte ich meinem Bruder Andrej Naumovski für seine bedingungslose Unterstützung bei dieser Recherche danken. Dieser Erfolg widmet sich ihm.

Acknowledgements

I would like to thank and express my gratitude towards Ao.univ.Prof. Dr. Andreas Rauber for the opportunity,guidance and support to work on this thesis.

I would like to express my gratitude towards Dipl. Ing. Thomas Lidy for his help explaining me the Feature Extractors and Psycho-acoustic Transformations for Music Genre Classification process. His time, dedication and help is truly appreciated.

I am also grateful to Guillermo Herrero Igeño and Damjan Gerovski for reading through the first draft and guiding me in the writing process.

I would like to thank my friends Nikola Neloski, Magdalena Piponska, Ljuben Necevki Iva Shokoska i Iva Nikolic for their love,support and understanding throughout this process.

The biggest gratitude goes to my parents for their unconditional love, faith, support and believe in me. Without them,this recognition would be impossible. My accomplishments are always their accomplishments as well.

Finally I would like to thank my brother Andrej Naumovski for his unconditional support during this research. This success is dedicate to him.

Kurzfassung

Die Informatik ist sehr jung und eine kontinuierlich fortschreitende Disziplin. Die Validierung und die Wirksamkeit einer wissenschaftlichen Forschung können neben ihren theoretischen Eigenschaften und Methoden oft durch Experimente validiert werden. Wissenschaftler und Studenten konfrontieren die Herausforderung, nicht in der Lage, die gleichen Ergebnisse bei der Wiedergabe von jemand anderes Experiment zu erreichen. Die meisten Rechenexperimente sind schlecht dokumentiert, experimentelle Ergebnisse sind kaum beschrieben und der Quellcode, der die Ergebnisse produzierte, ist selten verfügbar. Aus diesem Grund ist die Sorge um die Reproduzierbarkeit einer wissenschaftlichen Forschung in letzter Zeit stetig gestiegen.

Diese Arbeit konzentriert sich auf die Analyse des Einflusses verschiedener Betriebssysteme und verschiedener Entwicklungsumgebungen, wenn es darum geht, maschinelle Lernergebnisse zu reproduzieren. Darüber hinaus präsentiert diese Arbeit die Vor-und Nachteile der Reproduktion Maschine Lernalgorithmen auf verschiedene Version von Betriebssystemen und Entwicklungsumgebungen.

Die These kommt zu dem Schluss, dass der Wissenschaftler bei der Reproduktion eines Algorithmus für die Lernklassifizierung der Algorithmen sehr vorsichtig sein muss, wenn die ursprüngliche Entwicklungsumgebung oder das Betriebssystem geändert wird, da die Anzahl der in dieser Arbeit analysierten Experimente zeigt, dass sich unterschiedliche Umgebungen oder Betriebssysteme ändern können Die Ergebnisse der Ausgangsvorhersage des Modells.

Abstract

Computer science is very young and is a constantly progressing discipline.The validation and the effectiveness of a scientific research, besides its theoretical properties and methods, can often be validated by experimentation. Scientist and students confront the challenge of not being able to achieve the same results when reproducing someone else's experiment. Most computational experiments are poorly documented, experimental results are barely described and the source code that produced the results is rarely available. Because of this, the concern about reproducibility of a scientific research has been steadily rising recently.

This thesis focuses on analyzing the influence of different operating systems and different development environment when trying to reproduce machine learning results. Furthermore, this thesis presents the pros and cons of reproducing machine learning algorithms on different version of operating systems and development environments.

The thesis concludes that while reproducing a machine learning classification algorithm, the scientist need to be very cautious when changing the original development environment or operating system, due to the fact the number of experiments analyzed in this thesis show that different environments or operating system can change the results of the output prediction of the model.

Contents

CHAPTER 1

Introduction

1.1 Motivation

Computer science is very young and is a constantly progressing discipline. The scientific research focusing on computer science is progressing rapidly. Much of computer science is about expanding the usage of our current tools and algorithms rather than creating new ones.[BJ96] [Har95] The scientific research in computer science extends into a very wide sector of activities such as: information retrieval, image processing, artificial intelligence agents, machine learning and so on. The latest scientific achievements are based on previous ones.

The validation and the effectiveness of a scientific research, besides its theoretical properties and methods, can often be validated by experimentation. Scientist and students confront the challenge of not being able to achieve the same results when reproducing someone else's experiment. Most computational experiments are poorly documented, experimental results are barely described and the source code that produced the results is rarely available. Because of this, the concern about reproducibility of scientific research has been steadily rising recently.

As the movement to analyze and improve the reliability of a research progress, it is important to state that the term reproducibility is not standardized.[GFI16] There have been several attempts in order to define and distinguish the words "repeatability" and "replicability". De Roure [Rou14] stated that reproducibility means reusing a research objective with a different environments, circumstances and input data with a purpose to explore if the results are achieved independent of those changes. Also, the U.S. National Science Foundation (NSF) subcommittee on replicability in science [CKK+15] stated the following: "reproducibility refers to the ability of a researcher to duplicate the results of a prior study using the same materials as were used by the original investigator. That is, a second researcher might use the same raw data to build the same analysis files

and implement the same statistical analysis in an attempt to yield the same results.... Reproducibility is a minimum necessary condition for a finding to be believable and informative."

Results that can be reproducible are very beneficial not only for the people that will use them afterwards, but also for the scientist who obtained them as well.In order to achieve these reproducible results it is required that the researcher elaborates an execution pathway. Later on this pathway can be traced back easily. It also helps students and future researchers get familiar with the problem and tools that were used. With this being said, we can say that reproducibility is not only a moral responsibility with respect to the scientific field, but a lack of reproducibility can also be a burden for the individual researcher. [SNTH13]

When we talk about a computer science study being reproducible there is more than a work-flow that needs to be provided. Documenting reproducible computer scientific novelty implies keeping track of the following: an analytical dataset(original and processed data),the input parameters, the methods that were used and the computational environment where the experiments were performed.Every single change that we make to the previous points while reproducing a research will grant us with a different outcome and insight about its determinism, robustness, correctness and efficiency.[FFR16]

Nowadays, research studies based on Machine Learning and Deep Learning algorithms are thriving in the data science community. Knowing this, the reproducibility factor of those researches acquires a high relevance. This thesis will explain what happens with the results of scientific studies based on machine learning algorithms when they are reproduced on different environments and platforms and also using different methods. This thesis will as well clarify how the outputs are influenced when changing those parts. Showing the different reproduced results is a considerable contribution to the data science community.

1.2 Problem statement

The goal of thesis is to analyze and develop experiments focused on the challenges that come with reproducing a study mostly based on machine learning algorithms. As mentioned before, when reproducing any kind of research it is inevitable to change any of its original characteristics, such as: changing the environment,the input data,the research objective,the implementation or the parameters etc.As an additional goal, the thesis will analyze the impact of these changes to the result of the output when reproducing machine learning results.

All of the changes that can happen by reproducing a scientific research were structured in a model called PRIMAD [FFR16]. In this model the authors have attempted to categorize and label the various types of reproducibility by varying the (P)latform, (R)esearch Objective, (I)mplementation, (M)ethod, (A)ctor and (D)ata, analyzing the gain they bring to computational experiments. This thesis will be built upon some aspects on the

PRIMAD model, as changing the Implementation, Platform and Operating System in reproducing a machine learning results, by keeping the Hardware and the input data the same. The aim is to discuss the influence of different platforms and their versions run on different operating systems on reproducibility of machine learning results.

To analyze the outcome of the machine learning results in different circumstances, a number of machine learning experiments are implemented, whose end goal is to catalog a specific music file and associate it with its proper music genre. To extract features from the music files and create a data input file,the Feature Extractors and Psycho-acoustic Transformations for Music Genre Classification approach [LR05] is used. Using the features that are extracted from a music file, a "csv" file is created, which is used in every experiment as an input data.

Finding a specific music genre is the type of challenge that machine learning algorithms should be able to solve. Based on that argument, in this thesis three of the most used machine learning classification algorithms are tested: Support Vector Machine (SVM), Naive Bayes and Random Forest.The music information retrieval algorithms together with the machine learning algorithms are essentially implemented in Python 2.7 under Windows 10 operating system.

The first sub problem covered in this master thesis is analyzing the influence of different development environments such as Python,R,RapidMiner and Weka while reproducing machine learning algorithms. Implementing all of the machine learning algorithms in different environments requires changing the dependencies and the libraries used in each environment respectively. Also, checking the compatibility of the libraries used for each machine learning model and their referencing literature are crucial for the validity of the experiments.

A follow up experiment is created where the models are applied to the different versions of the implementations mentioned above. For some of the implementations, changing its respective version requires making an adaptation to the code flow. Each of the experimental models are adopted accordingly.

The second sub problem covered in this master thesis,is creating a number of experiments where each of the machine learning algorithms are tested under different operating systems (Windows,Mac OS and Linux) together with their different operating systems versions. The main requirements for this experiments are installing and preparing each of the operating systems to run on different virtual machines, with the same Control process unit (CPU) parameters and the same code and input data.

The main **c**hallenges that this project faced were:

1. The comparison of the effects that different inside processes (file systems,random number generator etc.) have in different operating systems.

2. Setting up the same parameters for different models and different machine learning libraries in different implementation environments.

3. Access to cross validation algorithm code in some of the implementation environments
4. Re-coding the models for different implementations and their respective requirements.

One of the challenges of this project is to find the reason behind different results depending on the operating systems, as well as to evaluate the flexibility and efficiency of a model.The reason behind this is the the fact that different operating systems such as Windows and Linux have different inside processes that cannot be compared easily(for example filling system).

The challenges related to the need of reproducing a machine learning model with all of the same parameters and input data,without the ability to trace some of the implementations environments such RapidMiner and Weka is usually more difficult than computing in a restriction-less environment. It is required to set up the same parameters to each of the experiments in order to ensure accurate and comparable results. The environments mentioned above that provide a Graphical user interface (GUI) are a challenge when it comes to setting up the right parameters in order to train the models.

In order to compare the performance of machine learning models, we are using a cross validation technique. The biggest challenge that we are dealing with is the lack of access to the code of a cross validation algorithm when using some of the environments.This results in some difficulties when trying to explain the different prediction outcomes in some of the cases.

As platforms progress during the years the coding standards from one version to another differ. In order to produce an experiment in different python versions such as Python 2 and Python 3, different coding standards are required. This results in the need to re-write all the models in order to fit the different versions of a platform.

1.3 Aim of the work

The main purpose of this thesis was to deliver detailed analysis and explanations of the different outputs that are caused by changing the development environment and operating system when reproducing a machine learning scientific experiment.

Furthermore, the analysis of the experiments done in this thesis will show the stability and flexibility of machine learning models under different circumstances.This approach opens a lot of questions and issues regarding the structure of the reproducibility resources that a scientist need to provide to make their novelty reproducible. Confidently, the results of this thesis will give valuable input to the reproducibility topic within the community, and help in its further development.

It is worth mentioning that the focus of this study is not finding the best machine learning algorithm for a genre classification problem, or finding the best parameters for each algorithm.This study, will focus on answering the following research questions:

1. What are the changes that happen when trying to reproduce a study using machine learning algorithm on different implementation platforms (Python/Weka/R/RapidMiner) and on different implementation platforms versions (Python 2.7/Python 3.6, Weka API 3.6/Weka API 3.7/Weka API 3.8, Weka 3.6/Weka 3.7/Weka 3.8)?
2. Which problems arise when we run the same machine learning algorithms and the same implementation environment on a different operative systems and different operative systems versions (Linux VS Mac OS VS Windows 7 VS Windows 8 VS Windows 10)?

This research also encourages for further and deeper analysis and experimentation into the machine learning community.Analyzing and experimenting deep learning algorithms is recognized as a next step for further analyses in this area. This thesis briefly covers this part and provides some basic experimentation models and analysis in the following chapters.

1.4 Methodological approach

The overall methodological approaches central to this thesis were mainly experimental and quantitative. The thesis began with an exhausting research, followed by the creation of a detailed experimental scheme. This experimental scheme was the ground for the future development of the thesis. Each experimental scheme represented a type of experimentation which analyses the behavior of machine learning models by changing certain circumstances (operating system and environment tools). For each of the experimental schemes the same output was expected - classifying music files into a specific music genre.

In each scheme 3 different machine learning classification algorithms were tested:

1. Support Vector Machine (SVM)
2. Naive Bayes
3. Random Forest

The first experimental scheme was used to explore and analyze the influence and the flexibility of different environment tools used in data mining and the influence that they have on the accuracy of the results of the machine learning algorithms. This thesis took into consideration two type of tools, script tools such as Python, R, WekaAPI and tools based on GUI such as: RapidMinder and Weka. Each of them required different code structure and library dependencies, but the input parameters and data remain the same. Furthermore, in this experimental scheme different versions of the above mentioned tools were used. We analyzed the two different version categories in Python (Python 2 and Python 3), also went deep into analyzing and comparing results using Weka 3.6, Weka 3.7 and Weka 3.8. Summarizing the first experimentation scheme, the different tools that were used were the following:

1. Python 2/Python 3
2. Weka 3.6/Weka 3.7/Weka 3.8
3. Weka 3.6 java API/Weka 3.7 Java API/Weka 3.8 Java API
4. R Studio
5. Rapid Miner 7.2

The second experimental scheme focused on analyzing the accuracy from all of the three machine learning algorithms when they are performed on different operating systems. For this experiment three different operating systems were explored:

1. Windows
2. Linux
3. Mac OS

In order to compare the results, and come to some conclusion, this experiment is done using the same three machine learning algorithms using the same parameters, input file (available at `https://zenodo.org/record/375815`) and code,while testing the influence of the operating system using all of the development environments (Python 2.7, Weka Application programming interface. (API) 3.7 , Weka GUI 3.7, R, RapidMiner).

Furthermore we explored the different outcome of the machine learning algorithms by performing the same input data and code,same parameters and same environment tool on different versions and families of the previous mentioned operating systems. In this scheme we compared the results from the following:

1. Windows 10
2. Windows 8
3. Windows 7
4. Linux Ubuntu 16
5. Linux Debian 8.6
6. Mac OS Sierra

Each one of the operating system was used as a virtual machine on the same hardware using Windows 10 as a base operating system.

The experimental stack is represented in Figure 1.1. As we can see from both experimental schemes explained above, we created and evaluated 30 different experiments in the first

experimental scheme and 180 different experiments in the second experimental scheme. A total of 210 experiments were made in order to analyze the influence of the operating systems and development environments to the machine learning results.

From all of the experiments mentioned above, we gathered the accuracy precision and recall measured results from all of the machine learning algorithms and performed a statistical analysis.

1.5 Structure of the work

In section 1 [Introduction], the problem and the necessity for a solution were defined which served as a motivation for this work. The goal of the thesis was stated and the methodological approach was outlined. In section 2 [State of the art], similar problems were analyzed, along with studies that have tried to address this type of problems before. Their differences and the reasons why they do not solve the problem at hand were noted.

Section 3 [Methodology] had the aim to address each experimental scheme in details. In this chapter the concepts,tools and experiments are explained. We start by explaining the experiments,doing detail analysis of the dataset used for analyzing the performance of machine learning algorithms. A brief introduction and explanation for each algorithm is provided in the methods and models section of this chapter. Next, the development environments together with the libraries which are used in each of them are stated in the Environmental section. In the last two parts of the Methodology chapter the second experimental scheme is discussed and the analysis methods that are used in evaluating the results are determined.

The fourth section [Results] is dedicated in explaining and analyzing all of the results gathered from the first and second experimental scheme. In each of them the problems that occurred are structured and retraced.

The last chapter is dedicated for summarizing and giving a critical reflection upon this thesis, followed by a discussion and suggestion for future work steps that can be taken for better progress in reproducibility work dedicated in reproducing machine learning algorithms.

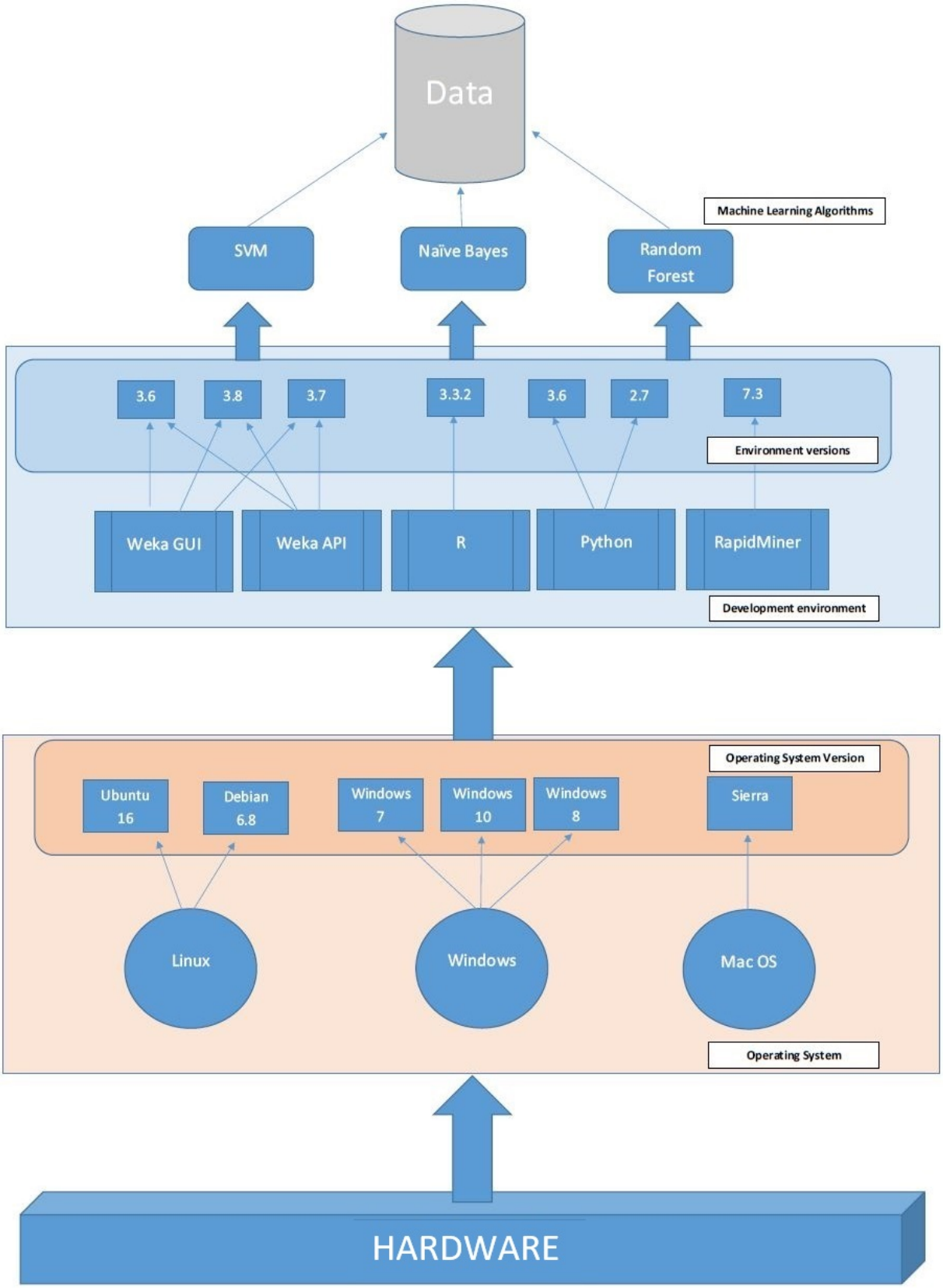

Figure 1.1: Hierarchical stack structure of all experiments run in this thesis.

CHAPTER 2

State of the art

2.1 Literature studies

There are many studies that aim to examine the reproducibility as a problem in the research community. In this section we will give a brief overview of each of the studies that had a big impact on creating this thesis. Analyzing and comparing them at the end of this chapter will help form a perspective of where the reproducibility community stands for today.

The first scientist that pointed the importance of reproducibility in science was the Irish chemist *Robert Boyle*, in England in the 17th century. As stated in the book Leviathan and the Air-Pump [SS11],historians of Science Steven Shapin and Simon Schafer describe in detail the debate between Boyle and Hobbes over the nature of vacuum. This debate gives the foundation of the argument of how important is the gain of knowledge. Boyle a pioneer of the experimental method, argued that the origin of knowledge should be empowered by facts which are experimentally proven by their reproducibility, in order for them to be scientifically acceptable. Furthermore, Boyle clarified that by repeating experiments over and over again, the certainty of fact will emerge.[1] Centuries later a philosopher of science*Karl Popper* stated briefly in his famous 1934 book "The Logic od Scientific Discovery"[Pop05] that "non-reproducible single occurrences are of no significance to science."

The Statistician *Rober Fisher* who set the standards for the nowadays scientific practice of hypothesis testings and statistical significance in 1935 in his famous book "The Design of Experiments" [FFG+60] wrote that "we may say that a phenomenon is experimentally demonstrable when we know how to conduct an experiment which will rarely fail to give us statistically significant results".All of this declarations suggest a common dogma in

[1] https://en.wikipedia.org/wiki/Reproducibility

modern science that reproducibility is a crucial condition for confirming a scientific fact, and for establishing authority in any field of knowledge in the scientific community.

A journal article was published in 2011 [Pen11] which talks about reproducible Research in Computer Science, it digs deeper into the physiology and why reproducibility is important for the computer science community. This paper ends with: "Ultimately, developing a culture of reproducibility in which it currently does not exist will require time and sustained effort from the scientific community." To be able to create new standards in any community, basic definitions need to be specifically defined. Defining reproducibility is the main issue that the scientist are dealing today. Finding the right structure by which one scientific novelty can be called reproducible has been a challenge.

The biggest issue that the computer science community is dealing with today is the fact that most of the scientific research papers do not support their findings with data or additional information. In 2007 Kovacevic [Kov07] explored 15 papers published in the *IEEE Transactions on Image Processing*. After Kovacevic read the papers and created a rating system for all of them depending on how well algorithms were explained and whether the code and data were available, she came to the conclusion that even if all the algorithms had proofs, none of the papers had their code available.Only 33% of the papers had their data available. In 2009 Vandewalle repeated the previous study from Kovacevic using larger sample size of 134 papers published in *IEEE Transactions on Image Processing* in 2004. Each of this papers was evaluated on its reproducibility by two or three reviews. The findings were not that different from the previous work. They stated that : " code (9%) and data (33%) are only available online in a minority of the cases, with data being available more often thanks to the frequent use of standard image data sets, such as Lena."

Referencing the two papers stated above,[CP16] in 2016 in the paper "Repeatability in computer systems research" Collberg, implements a study analyzing 613 papers and larger number of journals and conferences (13 all together) with an aim to analyze the existence of the code and data for each of this papers. The analysis showed that only 150 of all of the scientific papers had their code presented in the paper or on the publisher web-site. To have broader analyzes, Collberg contacted the authors of the studies she used, in cases when the data was not provided. She came to the result that only 22% of the scientists were open for sharing their code and information regarding their work. From all of the code data she gathered, 44% of them were able to run properly. With this research article, Collberg discussed the absence of available reproducible data in the computer science research community.

There are many papers that describe the purpose of reproducibility and try to find a solution of how to improve the percentage of reproducible scientific papers. In the paper "Reproducible Research in Computational Science" Roger D.Peng [Pen11] suggests a couple of steps that scientists need to take in order to produce a reproducible scientific research such as: publishing the code used for the research along with the data sets in a durable non-proprietary format, publishing the environment and the libraries used. Lastly D.Peng proposes creating a scientific community pool for storing all of the data,

code and meta-data and linking them to the publication, with a purpose of providing a single place to which people in all fields could turn to make their work reproducible. Willing to dig deeper into the problem of reproducing scientific work, Peng created a study providing a new approach of dealing with distributed reproducible research using cached computation.[PE09].In this paper he argues that "The replication of scientific findings using independent investigators, methods, data, equipment, and protocols has long been, and will continue to be, the standard by which scientific claims are evaluated. However, in many fields of study there are examples of scientific investigations that cannot be fully replicated because of a lack of time or resources. In such a situation, there is a need for a minimum standard that can fill the void between full replication and nothing. One candidate for this minimum standard is “reproducible research”, which requires that data sets and computer code be made available to others for verifying published results and conducting alternative analyses." In his work.[PE09] he describes a simple framework in which reproducible research can be conducted and distributed via cached computation. To use "cached computation" for distributing reproducible research, Peng created an add-on package for the R statistical commutating environment called "cacher". It provides tools for "caching" statistical analysis and distributing them to others in an efficient way. This package provides the authors and the readers tools for published statistical analysis.

De Rource in his paper "The future of scholarly communications.Insights." [Rou14] discussing the importance that scholarly communication, goes deeper into representing the articles as social objects which scholars share,cite and discuss. The *scholarly social machine* represented in this study, has been a great subject of discussion and engagement. The main aim of the scholarly communication is the fact that provides a platform for scholars to create, discuss and share information in social and research network, with a goal to measure their own reputation. This is the reason why reproducibility is from a very big importance in the research community. By having a *scholarly social machine* as proposed by De Rource, scholars need to provide a solid reproducible research which can be evaluated by the others.

There are studies that dig deeper into reproducibility of machine learning models with an aim to provide a standards for sharing reproducible research inside their community. In the paper "Towards Reproducible Descriptions of Neuronal Network Models", [NGP09] the authors stress out the problem of lacking a clear and common understanding of the role of computational models in neuro-science, as well as concluded practices that scientists need to have for describing network model in their publications. All 14 different research papers proposing neuronal network models were analyzed by their detailed description and the ordering and placement of their materials. Using this approach the authors proposed a "good model description practice" consisting of guidelines for the organization of publication, a checklist for model descriptions, templates for tables presenting model structure and guidelines for diagrams of networks.

In 2012, a group of scientists investigated the effect that different FreeSurfer versions, workstation types and Macintosh Operating System Versions have on anatomical volume

and cortical thickness measurement [GHJ+12]. Even if this study was done in the field of the Neurophysiology and Neuroscience, this PLOSOne paper on Freesurfer is being the most relevant analysis / most similar study on the impact of OS and implementations. This study shows a significant difference between FreeSurfer version 5.0.0 and two earlier versions on calculating the anatomical volume and cortical measurement. Also, the researchers detected small differences between Macintosh and Hewlett-Packard workstation and between different versions of operating systems. As they state "In neuroimaging, due to the variations of software and/or libraries, the execution site often has to be controlled by the users to guarantee the correctness and reproducibility of computations.

The idea for this thesis came from a model that was created in the Dagstuhl Seminar on "Reproducibility of Data-Oriented Experiments in e-Science"[FFR16], held in Dagstuhl, Germany, from the 24th to the 29th of January of 2016. A group of scientists gathered to discuss and share their knowledge from different fields of computer science – databases, information retrieval, information visualization, semantics, bioinformatics, human-computer interaction, simulation, and more, with a ultimate goal of creating a common ground across disciplines, leverage best-of-field approaches, and provide a unifying vision on reproducibility. By gathering their knowledge and experience the scientists created a model PRIMAD with an aim to tackle with the different angles of reproducibility. The model PRIMAD was made from the first letters of Platform, Research goal, Implementation, Method, Actor, and Data which as the scientists say " the main circumstances that can be changed while reproducing a study.". By changing any of this "variables", we are able to gain additional information regarding a specific research. Below, examples of different benefit that the PRIMAD model brings to a reproducible scientific research is presented.

This thesis will not focus on creating a model for reproducible machine learning models as the authors did in the previous discussed work. Nevertheless we will examine and analyze the reaction of specific machine learning algorithms run under different operating systems, implemented in different development environments.

2.2 Comparison and summary of existing approaches

As discussed in the previous section, there are many studies that focus on reproducibility in general nowadays. Many of them focus on creating a methodological structure of developing more reproducible research papers. The biggest problem that computer scientists researching this topic find is the low number of studies which have their code accessible. Another problem that plays a big role in reproducing a computer science study is the lack of detail code documentation and errors which usually occur when a second scientist is trying to reproduce the same published code.

In machine learning community, many researchers focus on creating standards of sharing reproducible research inside their community. The problem that occurs when dealing

Label	Data		Platform / Stack	Implementation	Method	Research Objective	Actor	Gain
	Parameters	Raw Data						
Repeat	-	-	-	-	-	-		Determinism
Param. Sweep	x	-	-	-	-	-		Robustness / Sensitivity
Generalize	(x)	x	-	-	-	-		Applicability across different settings
Port	-	-	x	-	-	-		Portability across platforms, flexibility
Re-code	-	-	(x)	x	-	-		Correctness of implementation, flexibility, adoption, efficiency
Validate	(x)	(x)	(x)	(x)	x	-		Correctness of hypothesis, validation via different approach
Re-use	-	-	-	-	-	x		Apply code in different settings, Re-purpose
Independent *x* *(orthogonal)*							x	Sufficiency of information, independent verification

Figure 2.1: PRIMAD Model: Categorizing the various types of reproducibility by varying the (P)latform, (R)esearch Objective, (I)mplementation, (M)ethod, (A)ctor and (D)ata, analyzing the gain they bring to computational experiments. x denotes the variable primed i.e. changed, (x) a variable that may need to be changed as a consequence, whereas - denotes no change. Figure taken from the Reproducibility of Data-Oriented Experiments in e-Science[FFR16]

with this issue is the lack of clear and common understanding of the role of computational models.

By creating the Dagstuhl Seminar on “Reproducibility of Data-Oriented Experiments in e-Science" scientists were interested to analyze the influence of other environments when trying to reproduce a research. This analysis as they state is for big importance for the further development on reproducibility standards in the future.

This master thesis explored a core part of the PRIMAD models such as changing the platform and the implementation via keeping the research objectives, the methods and the data in reproducing machine learning results. We hope that the analysis coming from this study will open new questions for further research in the reproducibility community.

CHAPTER 3

Methodology

In this chapter, we will describe in detail the methodology and concepts that are used in this master thesis. To be able to analyze the influence that different development environments and the operating systems have on machine learning results, two types of experiment schemes are created. In order to resolve the flexibility,adoption and the efficiency on reproducing machine learning results the first experimental scheme focused on analyzing the effect of the development environment run on a fixed operating system. On the other hand, analyzing the validation of hypothesis and the correctness of implementation was the goal of the second scheme which includes experiments developed in a fixed environment, run on different operating systems. The schemes were split in two for better structure of this work.

This chapter begins with an overview of the used concepts included in this thesis. The used concepts section includes a description of the source and the algorithms behind the Psycho-acoustic Transformations for Music Genre Classification approach [LR05] which is used as a feature extractor of the GTZAN database. Furthermore, an overview and a definition of machine learning models, training machine learning models and cross validation is included.

All of the experiments follow the same experimental flow as shown in Figure 3.1.Each experiment uses the same data as an input. The input data (csv file) contains 168 features for each audio file together with its genre. After the data input, three different machine learning classification algorithms are trained. An exhaustive description and explanation of the decision to chose specifically those machine learning algorithms, detail overview of them and their parameters are a topic of discussion in the third chapter of study.Each machine learning algorithm is evaluated by cross validation, from which the results are analyzed for each experimental scheme separately. The cross validation allow us to use the entire dataset to train and test one model/method,while being able to have a reasonable idea of how well it will generalize. On the other hand the cross validation is mostly used in any research based on machine learning algorithms because of its tendency

to do a model selection on the model parameters. The details about which environments are used in the first experimental scheme and their setup is detailed explained in the fourth section. The fifth section discusses the different operating systems that are used for the second experimental scheme. The last section of this Chapter focuses on the analyzed methods that are used to gather and conclude the results for each experimental scheme separately.

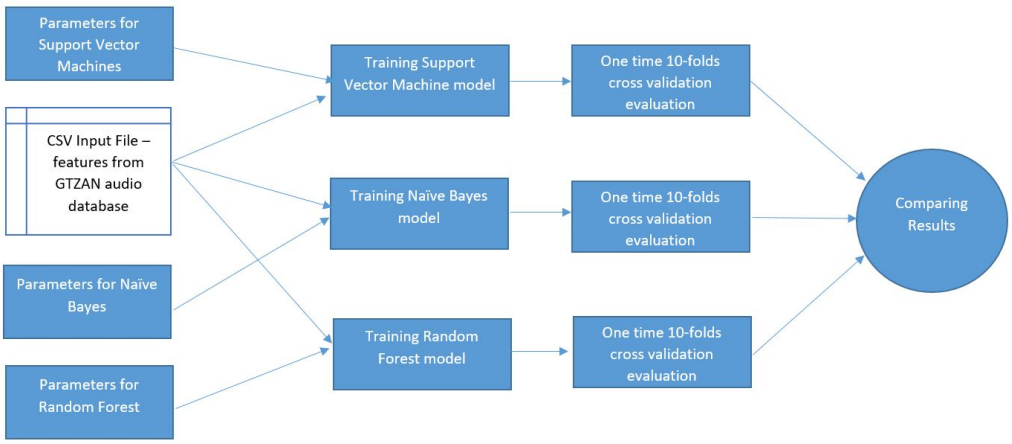

Figure 3.1: This Figure represents the experimental structure and flow followed by both experimental schemes. Each experiment takes the following steps: Takes an "csv" file with 169 features from GTZAN audio database as an input, set parameters for a machine learning algorithm and train and cross validate the model. Based on the aim of the experiment, each of the steps were developed in a specifically defined environment and run on a specific operating system. The results were compared at the end for each experimental scheme.

3.1 Used concepts

In this section, several predefined concepts which were used in this master thesis will be discussed. This concepts are not crucial for the end result and the conclusion of this study, but are necessary for the experimental establishment. We will mainly focus on describing the Psycho-acoustic Transformations for Music Genre Classification approach [LR05], and the concepts used for training and evaluating any machine learning model such as: parameter settings,training a model and cross validation evaluation. Each of them is described in a separate subsection and an overview of their importance is provided at the end.

3.1.1 Feature Extraction -Psycho-acoustic Transformations for Music Genre Classification

Feature Extraction is the essence of content-based description of audio files. By extracting features from audio files, we enable the computer to recognize the content of a piece of music without the need to define additional labels such as: artists, genre or song title. Content-based description requires the development of feature extraction techniques that analyze the acoustic characteristics of the signal. Features extracted from the audio signal are intended to describe the stylistic content of the music, e.g. beat, presence of voice, timbre, etc.

In 2005 Lidy and Rauber published a paper " Evaluation of feature extractors and psycho-acoustic transformations for music genre classification"[LR05] with an aim to extract suitable semantic information from music. They used methods from digital signal processing and consider psycho-acoustic models to develop additional feature sets from audio files to extract feature informations for musical genre classification.The Figure 3.2 presented in " Evaluation of feature extractors and psycho-acoustic transformations for music genre classification" paper [LR05], explain all of the processes and algorithmic steps that were taken to create the extraction of features from audio files.

The feature set that will be used in this master thesis is called SSD(Statistical Spectrum Descriptor)After couple of preprocessing steps on a input audio dataset,the algorithm presented, implements a transformation of audio segment into spectrogram representation using Fast Fourier Transformation (FFT) with hanning window function (23 ms windows) and 50 %. On this data a Bark scale was applied for grouping frequency bands into 24 critical bands. The transformation to 24 critical bands represents rhythmic characteristics within the specific frequency range of a critical band. For all of the 24 critical bands during feature extraction step, a Statistical Spectrum Descriptor SSD was computed. The SSD features" describe the rhythmic content of a piece of audio by computing the following statistical moments on the values of each of the 24 critical bands: mean, median, variance, skewness, kurtosis, min- and max-value."

The code for this algorithm[1] is reused for the feature extraction process from audio files in this master thesis. The algorithm for extracting SSD features returns a list of 168 feature attributes for each audio file in a csv format. This file is used as an input file for all of the experiments evaluated in this study. The Feature Extraction-Psycho-acoustic Transformation task is not relevant for the end results in this thesis.

3.1.2 Machine Learning concepts

Machine Learning is the science behind getting the computers performing a specific task without being especially programmed for it. There are multiple machine learning techniques available nowadays. This Master Thesis focuses on classification of audio

[1] http://ifs.tuwien.ac.at/mir/downloads.html

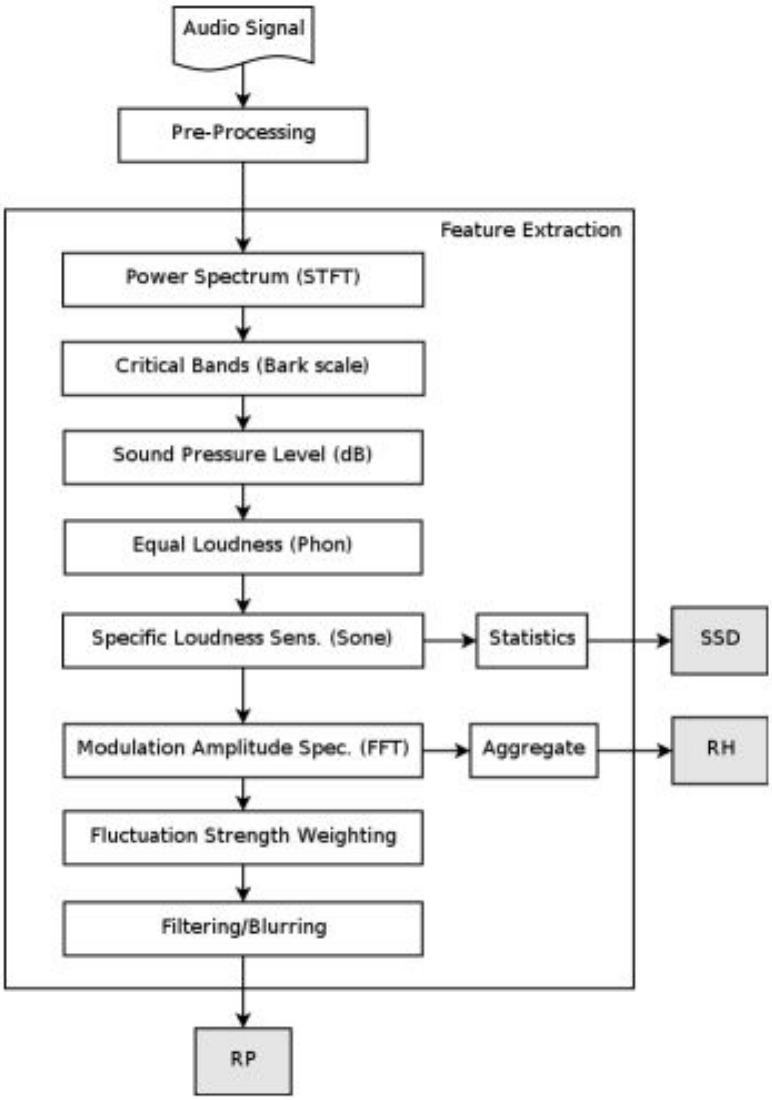

Figure 3.2: Block diagram of audio feature extraction

music data into different nominal categories, witch requires using supervised classification machine learning algorithms.

Training a machine learning model

Training a machine learning model represents applying a specific machine learning algorithm with specific parameters to a set of training data. The training data needs to

represent a real world problem. It needs to provide to the machine learning algorithm a set of features that describe the problem that needs to be solved together with the expected output values or classes.

In this master thesis we train classification machine learning algorithms which require the test data to have output values for each sample in a nominal form(the output needs to be a class label).

Setting up right hyper-parameters for the chosen algorithm it is crucial in the training process.These parameters may be adjusted by optimizing performance on a subset (called a validation set) of the training set, or via cross-validation. This process is a usually a main focus in Machine Learning research. Regarding the fact this study focal point is on reproducibility on existing machine learning results, we will not focus on optimizing the hyper-parameters for each machine learning algorithms used in the experimental schemes.

Cross Validation

Testing the performance of a machine learning model after training, requires providing a test sample on which the evaluation of the model will be performed.The test sample needs to describe the same real world problem and have the same features attributes as the training set. The output of the model is compared with the true class labels of the training set.

Providing a test data for testing the performance on the machine learning model is not the only way to get accuracy results. The k-fold cross validation is a model validation technique that splits the training set into equal k smaller sets. For each of the k-"folds",a model is trained using k-1 of the folds as a training data, and the resulting model is validated on the remaining part of the data. The average of the values computed in the loop is the performance measure that is reported by the k-cross validation.

The tendency to do a model selection on the model parameters and being able to have a reasonable idea of how well the data is generalized are the main reasons why the cross validation technique is mostly used in the machine learning studies. As the aim of this thesis is to question the reproducibility of machine learning results, in our experiments we are using a one time 10-folds cross validation evaluation for evaluating the models.

3.2 Data models

3.2.1 GTZAN audio collection

The experiments are performed on a audio collection that was used by George Tzanetakis in his famous paper in genre classification " Musical genre classification of audio signals "[TC02],named GTZAN. The files in this collection were collected in 2000-2001 from a different type of sources such as:personal CDs, radio or microphone recordings. It

contains 1000 audio files equally distributed in 10 different music genres.For details about the genders involved and the destruction of the documents in each class refer to Table 3.1

GTZAN	1000
blues	100
classical	100
country	100
disco	100
hiphop	100
jazz	100
metal	100
pop	100
reggae	100
rock	100

Table 3.1: GTZAN audio collection listing classes and number of titles per class

3.2.2 Data after Feature Extraction

As discussed above the feature extraction was done by the "psycho-acoustic transformations for music genre classification" algorithm[LR05] providing csv file of SSD feature data for every audio file in the GTZAN audio collection. The file contained of 169 numerical features and one nominal class describing the music genre. All of the features are standardized to have zero mean and standard deviation of 1 (apart from the class attribute).

The Feature Extraction algorithm presented in [2] is run on Windows 10 (64-bit) operating system using Python 2.7 development environment. The data extracted from this algorithm is stored in a "csv" file [3] which is used as an input file throughout all experiments in this master thesis.

3.3 Methods and models

This section provides an overview of the machine learning algorithms and their parameters used in this master thesis. All of the machine learning algorithms are classification algorithms.

Classification task usually involves separating data into training and test set as discussed in the previous sections. Each instance of the training data consists of one class label ("target value") and several features of observed variables. For this thesis we chose to analyze three different types of classification algorithms such as linear classifiers (Support

[2] http://www.ifs.tuwien.ac.at/mir/webservice/

[3] Zenodo link for accessing the CSV file `https://zenodo.org/record/375815`

Vector Machine), Bayes classifier(Naive Bayes) and tree based classifier (Random Forest). Description and details for each of the classification algorithms used is given in the following sections.

3.3.1 Support Vector Machines

SVMs (Support Vector Machines) are one of the most used techniques for data classification. The goal of SVMs as all the others classification algorithms, is to provide a model(based on the training data) which predicts the class(target) values of the test data given the test data attributes.An Support Vector Machine model is representing the examples as points in space,mapped in a way that examples of the separate categories are divided as wide as possible. In order to predict the class of a new examples, Support vector machine model map the examples into the same space and predict the class based on which side of the gap they fall. Below some underlying theory will be provided, with a purpose of understanding the parameters needed to run a SVMs model.

Boser in 1992 in his article "A training algorithm for optimal margin classifiers" [BGV92] laid down the foundation of the theory behind SVMs. Given a training set of instance-label pairs $(x_i, y_j), i = 1....m$ where $x_i \in R^n$ and $y \in 1, -1^m$ Boser desired the solution to the following optimization problem:

$$\begin{aligned} min_{w,b,\xi} \frac{1}{2} w^T w + C \sum_{i=1}^{N} \xi_i \\ \text{subject to } y_i(w^T \phi(x_i + b) \geq 1 - \xi_i \\ \xi_i \geq 0, i = 0, .., N) \end{aligned} \tag{3.1}$$

Using the function ϕ, the SVMs algorithm maps the training vectors x_i into a higher(maybe infinite) dimensional space. SVM creates a linear separating hyperplane using the maximal margin in the higher dimensional space. w represents the vector of coefficients, b is a constant and ξ_i represents parameters for handling the non-separable data (inputs).The parameter C>0 represents the penalty error. The C parameter tells the SVM optimization how much avoiding misclassifying each training example is acceptable. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassified more points.

Boser with his work [BGV92] had the most influence in creating the so called kernel functions $K(x_i, x_j) \equiv \phi(x_i)^T \phi(x_j)$. Kernel functions operate in a high dimensional space. The coordinates of the data in that space are never computed by the Kernel functions, but a inner products between images of all pairs of data is computed in the feature space. There are many new kernel functions that come out in different research studies every day, but the tools that we use to run the experiments mostly include the following:

1. linear$K(x_i, x_j) = x_i^T x_j$
2. polynomial $K(x_i, x_j) = (\gamma x_i^T x_j + r)^d, \gamma > 0$
3. radial basis function (RBF): $K(x_i, x_j) = exp(-\gamma||x_i - x_j||^2), \gamma > 0$
4. sigmoid:$K(x_i, x_j) = tanh(\gamma x_i^T x_j + r)$

Besides the kernel function and the C value using the SVM in practice, there are many parameters that need to be set up and play an important role in the performance of the model. The most important ones are:

- Degree : The degree hyper-parameter is used only when a polynomial kernel is used. In all other cases is ignored.
- Gamma : This parameter is only used when kernel type is set to "polynomial","rfb" or "sigmoid". Changing the value of gamma is changing the accuracy of the resulting model.
- Cache size: Setting up the size of the kernel cache in Megabytes.
- Tol: Tolerance of stopping criterion.
- Random state: The seed of the pseudo random number generator to use when shuffling the data for probability estimation.
- Class weight: Represents parameter that gives emphasis to each class that need to be learn.The cost/penalty function will adjust accordingly to the cost in order to focus the learning on that class.

In 2005 a LibSVM software was created [CL11] with a purpose of helping users from different fields to use SVM. LibSVM provides a very simple inteface so that users can link it with their own program. LibSVM package is available in all of the big data mining languages and platforms. The LibSVN packages that will be used during this thesis are:

- Python : scikit-learn package with classes SVM (sklearn.svm.SVC) [4]
- R: Package e1071 - interface to libsvm for R usage [Dav15]
- Weka: LibSVM - wrapper class for the libsvm original library [5]
- RapidMiner: package libSVM - based on Java LibSVM implementation[6]

[4] http://scikit-learn.org/stable/modules/svm.html
[5] https://weka.wikispaces.com/LibSVMl
[6] *http : //docs.rapidminer.com/studio/operators/modeling/predictive/support*$_v$*ector*$_m$*achines/support*$_v$*ector*$_m$*achin*

More details about the implementation of the different packages in different software implementations will be discussed in the next chapter.

In oder to make the results of the analysis comparable with each other, all of the SVMs implementations need to have the same parameters for every experiment throughout the thesis as shown in the table 3.2 below.

SVMs parameters	values
SVMType	"C-SVC(classification)
C	1.0
kernel	"linear"
tol	0.001
cache size	200
random state	1
class weight	none

Table 3.2: List of set up parameter values for running SVM algorithm in every environment and language throughout the experiments

3.3.2 Naive Bayes Classifier

Naive Bayes classifiers belong to a family of simple probabilistic classifiers based on applying Bayes' theorem with a naive assumption between predictors. A Naive Bayes model is very easy to build. It doesn't require complicated iterative parameter estimation. Despite its simplicity, Naive Bayes often performs well and many times outperforms more sophisticated classification methods. As result of that, Naive Bayes is widely used nowadays.

3.3.3 Gaussian Naive Bayes

Naive Bayes can be extended to real-valued attributes. To do this, the most common practice is to assume a Gaussian distribution.

This extension of naive Bayes is called Gaussian Naive Bayes. The Gaussian Naive Bayes is the easiest function that can be used as an extension, only because it requires an estimation of the mean of the standard deviation of the training set.

The Naive Bayes model in its core, calculates the class (target) probabilities, by calculating the frequency of instances that belong to each class dividing it by the total number of instances. In addition to the probabilities for each class, the Gaussian Naive Bayes, stores the mean and the standard deviation for each input variable for each class.

Gaussian Naive Bayes Model makes prediction of new entry by calculating the "Gaussian Probability Density Function"(PDF). The PDF provides an estimate of the probability of a new entry for a specific class.

$$pdf(x, mean, sd) = (1/(\sqrt{2\pi})sd \cdot e^{(-\frac{x - mean^2}{2sd^2})} \tag{3.2}$$

The Gaussian Naive Bayes model, because of its simplicity, doesn't require setting up specific parameters. Different packages based on the Gaussian Naive Bayes model are provided for different data mining languages and tools. The packages used in this master thesis are the following:

- Python: The package "sklearn" provides a sub-package called GaussianNB.[7]
- R : The provides "caret" provides an extension in its train method for using Gaussian Naive Bayes classifier.The Naive Bayes method is using an additional package "klaR". [8]
- Weka: The package Naive Bayes stored in the package AbstractClassifier. [9]
- RapidMiner : Naive Bayes package from RapiMiner Studio Core. [10]

3.3.4 Random Forest Classifier

Random Forest Classifier is an ensemble learning method used for classification and regression. It operates by creating a number of decision trees during training time. Decision trees output a prediction class by creating a mode of the classes of the individual trees in classification cases. The most important parameters that are used in creating a Random Forest model are the following:

- Number of estimators: A hyper parameters describing the number of trees that the algorithm need to produce in the forest.
- Criterion: An parameter stating a function that measure the quality of a split. Some of the supported criteria are Gini impurity and the "entropy" for the information gain.
- Max depth: This hyper-parameter expresses the maximum depth of the tree.
- Number of features: A parameter setting up the number of attributes to be used in random selection.
- seed -The seed parameter sets the Random Number Generator.

In order to get comparable results from all of the Random Forest implementations we use specific values for all of the parameters shown in Table 3.3

[7] *http : //scikit − learn.org/stable/modules/generated/sklearn.naive*$_b$*ayes.GaussianNB.html*
[8] https://rdrr.io/rforge/caret/man/models.html
[9] http://weka.sourceforge.net/doc.dev/weka/classifiers/bayes/NaiveBayes.html
[10] *http : //docs.rapidminer.com/studio/operators/modeling/predictive/bayesian/naive*$_b$*ayes.html*

Random Forest parameters	values
Number of estimators	10
Criterion	"gini"
Max depth	15
Number of features	$\sqrt{\text{number of features}}$
Seed	15325

Table 3.3: List of set up parameter values for running Random Forest algorithm in every environment and language throughout the experiments

All the experiments throughout this master thesis are run in different environments. Each of the development environment has different packages that are performing the Random Forest algorithm. All of the used packages are given below:

- Python: The package "sklearn" provides an implementation of Random Forest algorithm [11]
- R : The R languages uses the package "randomForest" . [12]
- Weka: The package Random Forest stored in the package "Bagging". [13].
- RapidMiner: Random Forest package from "RapidMiner Studio Core". [14]

All of the above mentioned packages, use the algorithm implemented by Breiman in his work "Random Forest" [Bre01].

3.4 Environments

In this section the first experimental scheme will be provided in details. The aim of the experiments in the first experimental scheme is to analyze and discuss the influence that changing the development environments have during reproducibility of machine learning results.

Every experiment is done using the same Hardware running on the Windows 10 (64-bit) operating system. Due to the fact the aim of the first experimental scheme is to analyze the influence of development environment in machine learning results we implement Support Vector Machine, Naive Bayes and Random Forest in 5 different environments (R, RapidMiner, Weka GUI, Weka API and Python). Furthermore, we test the influence of

[11] $http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html$
[12] $https://cran.r-project.org/web/packages/randomForest/randomForest.pdf$
[13] $http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/RandomForest.html$
[14] $http://docs.rapidminer.com/studio/operators/modeling/predictive/trees/parallel_random_forest.html$

different versions of the environments such as Weka GUI and Weka API 3.6,3.7 and 3.8 and also different Python versions such as Python 2.7 and Python 3.6. Each experiment is using the extracted features from the GTZAN audio collection database presented in a "csv" file. The csv file is available on this link `https://zenodo.org/record/375815`. All of the structure for the experiments from the first experimental scheme is shown in the Figure 3.3.

In this chapter we will describe the settings for each of the environments and their different versions separately in number of sub-sections.

3.4.1 Python

Python is a high-level, scripting, dynamic programming language. It allows programmers to write more structured code,and with few lines of code to express concepts that usually will take lot more effort in other programming languages.

In this master thesis, every machine learning model is implemented in Python 2 and Python 3 respectively. Each model is trained with specific parameters mentioned in the section above. The evaluation of the model is done by a one time 10-fold cross validation providing the accuracy of the model as a average of all of the accuracy from each fold using the Python package "skitlearn".

The Python implementation run using a package management systems Anaconda 2 [15] and pip [16] installed on a Windows 10 (64-bit) operating system. The packages that were used for experiments developed in Python are presented in Table 3.4

Packages	Version
numpy	1.11.2
scipy	0.18.1
argparse	1.2.1
unicsv	1.0.0
pandas	0.19.0
scikit-learn	0.18.1

Table 3.4: List of required packages for Python implementation.

The reading of the "csv" files, training and the cross validation evaluation is done using "read csv" [17], "cross validation" [18], "cross val predict" [19] and "confusion matrix" [20] packages inherited from skit-learn main package.

[15] https://www.continuum.io/downloads
[16] https://pypi.python.org/pypi/pip
[17] $https://docs.python.org/2/library/csv.html$
[18] $http://scikit-learn.org/stable/modules/cross_validation.html$
[19] $http://scikit-learn.org/stable/tutorial/statistical_inference/model_selection.html$
[20] $http://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html$

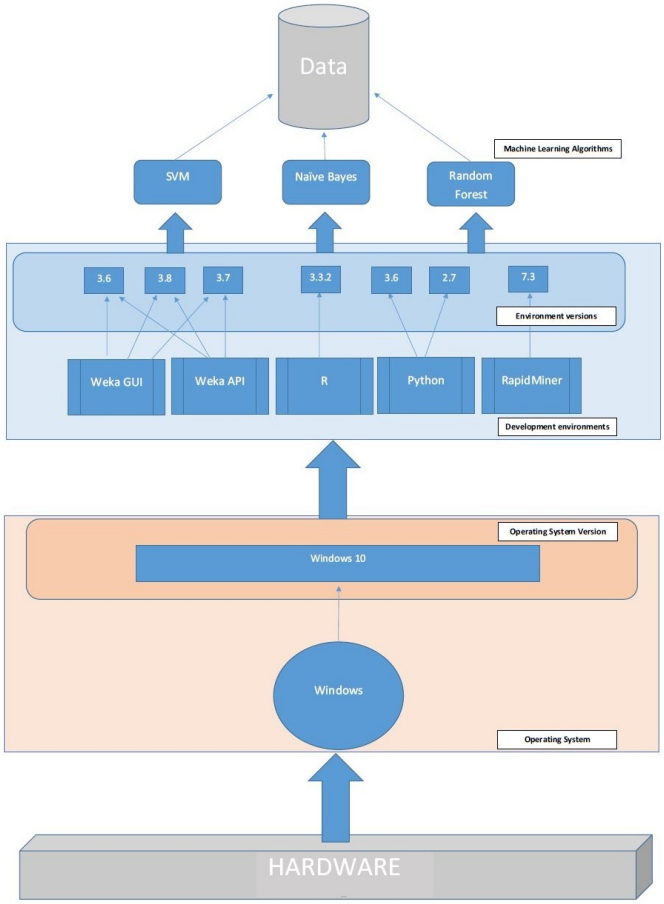

Figure 3.3: In this Figure the structure of the experiments in the first experimental scheme is represented. Every experiment was done using the same Hardware running on the Windows 10 operating system. Due to the fact the the aim of the first experimental scheme was to analyze the influence of development environment in machine learning algorithms we implemented Support Vector Machine, Naive Bayes and Random Forest in 5 different environments (R, RapidMiner, Weka GUI, Weka API and Python). Furthermore, we tested the influence of different versions of the environments such as Weka GUI and Weka API 3.6,3.7 and 3.8 and also different Python versions such as Python 2.7 and Python 3.6. Each experiment was using the same GTZAN audio collection database.

The Python implementation is developed in a way that it requires input parameters such as: a "csv" file input, a parameter stating if a cross-validate accuracy need to be performed on the input data and three different parameters stating one of the machine learning model that needs to be used (SVM,Naive Bayes or Random Forest). The most crucial part of the code implemented in Python is shown in a Figure **??** and Figure **??** in Appendix C.

Different versions of Python

The initial experiment developed in Python is using Python 2.7. Analyzing the difference that a version of development environment can bring to reproducing a machine learning algorithm, the same code needs to be adapted to fit the standards of Python 3.6 version.The Table 3.5 shows the difference between Python 2 and Python 3 implementation and all of the changes that needed to be done in order for the same code from Python 2.7 to be migrated in Python 3.6 implementation.

Python 2.7	Python 3.6	Description
print	print ()	print function
5/2 = 2	5/2 = 2.5	devision of two integer numbers
unicode	str	assign text data type
dict.values()	list(dict.values)	list of values from a dictionary

Table 3.5: Changes required in transferring Python 2.7 code to Python 3.6

Python versions 2.7 and 3.6 are using the same libraries and the same versions as presented in the Table 3.4.

3.4.2 R

R is a programming language and software environment for statistical computing supported by the R Fundation for Statistical Computing.[Hor12]. The R language and environment is mostly famous for its use among statistician and data miners for developing statistical software and data analysis.

Because of its popularity among scientists, in this master thesis the analysis of the R language influence of reproducibility of machine learning algorithms is taken into consideration. The R implementation is done in R Studio 1.0.44 under Windows 10 (64 bit) operating system.

To recreate the program that is done previously in Python, a "csv" file is imported in R including all of the feature attributes and classes for all of the audio files from the GTZAN audio collection. Furthermore, all of the machine learning models are developed using R libraries as mentioned in the section 3.3. The evaluation of the model is done using 10-cross validation with random seed of 1.The most important part of the implementation in R is shown in Appendix E.

3.4.3 Weka

Weka is a data mining software implemented in Java providing a collection of machine learning algorithms for data mining problems. Weka provides two different kind of usages of their algorithms, Weka graphical user interface that allows easy accesses and fast

results for data mining and an API for Java where all the Weka algorithms are provided to be used inside of a Java program.[21]

In this master thesis, two different types of experiments are done. In the first place, a Weka GUI tool is used to perform a music genre classification out of a GTZAN audio collection. Also, we want to research the stability of the Weka algorithms, by performing a second type of experiments with the same aim with using Weka API on Java 8 environment. For both of this experiments three different versions of Weka are tested: Weka 3.6, Weka 3,7 and Weka 3,8. In this section we will describe in details the processes taken for all experiments using Weka.

Experiments using Weka - GUI

In order to input the "csv" file a Weka "Knowledge Flow" CSV Loader operator is used. The "csv" file contains all of the features from the Feature Extraction step together with their respective classes for the audio files from GTZAN audio collection. The class assigner operator is used for assigning the class labels for every instance, by taking the last column from the database. Each one of the three different models is trained using Weka implementation of the algorithms as described in section 3.3. A "cross validation FoldMaker" operator is integrated after the classes assignments which provides 10 folds of test and training data used in each of the machine learning operators. The classifier Performance Evaluator operator is used to gather all of the evaluations from each model and extract via text using the Text Viewer operator. The implementation of all three machine learning models, together with its parameters are shown in the Figure 6.1 6.2,6.3 in Appendix A. To be able to analyze the structure of each fold we extract "csv" files including training and test sets for each fold iteration. For every model, all of the results from the 10-fold cross validation are saved for the the analysis of the variance of each fold in the cross validation split.

Expiriments using Weka API

Support Vector Machines, Naive Bayes and Random Forest algorithms are implemented using Weka packages in Java 8 environment. For developing this experiments, we are using Eclipse IDE[22] as an development environment. The Weka API is imported through jar files from the build path function in Eclipse.

At first,the program read an "arff" file containing the SSD features from a GTZAN audio collection. Furthermore, all of the machine learning algorithms are implemented using respective classes from the Weka package, and specific parameters are set up as discussed in section 3.3. To analyze the performance of the models, as in all of the development implementations a 10-fold cross validation is implemented with a random seed of 1 at the end. This code is run using Weka 3.6,Weka 3.7 and Weka 3.8 packages. In Appendix B the code of the implementation is provided.

[21] http://www.cs.waikato.ac.nz/ml/weka/

[22] http://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/keplersr1

Different versions of Weka

All of the algorithms are tested in different versions of Weka such as Weka 3.6, Weka 3,7 and Weka 3,8.Each version is downloaded from the official Weka site [23].

3.4.4 RapidMiner

RapidMiner is an open source platform developed in Java, that provides data mining, text mining, predictive analytics and business analytics. It is widely used by students and researchers because of its very user friendly API, that provides designing and execution of analytical flow from data. The workflows are called "Processes" in RapidMiner, and each process consists of multiple "Operators". Each operator performs a single task within the process. The operators are connected during a process, such that an output of each operator creates an input of the next one.

For our experiments we use Rapid Miner Studio 7.3, using RapidMiner implemented operators for each of the machine learning models. Each of the processes is retrieving the data set from a "csv" file, from where a SetRole operator is implemented. A Set Role operator is setting the class feature as a target label for the model. Connected to this operator is the operator which does a 10-fold cross validation operator using a Stratified Sampling with a random seed of 1. Every experiment implemented in RapidMiner had the same previously mentioned base as shown in Figure 6.4 in Appendix D.

Inside of the Cross Validation operator, a training side and a testing side are presented. The training side includes an operator representing a specific model that we want to train and the testing side includes the Apply Model and Performance operators witch give us the performance of the model for every fold. An example of a Random Forest implementation is shown in a Figure 6.5 included in the Appendix D. For implementation of Support Vector Machine and Naive Bayes algorithms only the operator in the training side was changed.

3.5 Operating systems

In this following section, the second experimental scheme will be provided in details. As we discussed in the motivation section of this thesis, the goal of the experiments in the second experimental scheme is to analyze the influence that operating systems have on the performance of machine learning algorithms during reproducibility. I

As shown in the Figure 3.4, Support Vector Machine, Naive Bayes and Random Forest algorithms were implemented in Python 2.7, RapidMiner, R, Weka 3.8 API and Weka 3.8 GUI using fixed parameters and fixed database. Using only one version of the presented development environments was enough to be able to analyze the hypothesis whether changing the operating system changes the performance results in reproducing machine learning algorithms. For each of the algorithms, a one time 10-cross validation was

[23] http://www.cs.waikato.ac.nz/ml/weka/downloading.html

used for evaluating the models. The Each environment was run in Linux, Windows and Mac operating systems. Different versions of the operating systems were used such as Windows 10(64-bit), Windows 8 (64-bit), Windows 7(64-bit), Linux Ubuntu 16, Linux Debian 6.8 and Mac OS Sierra.

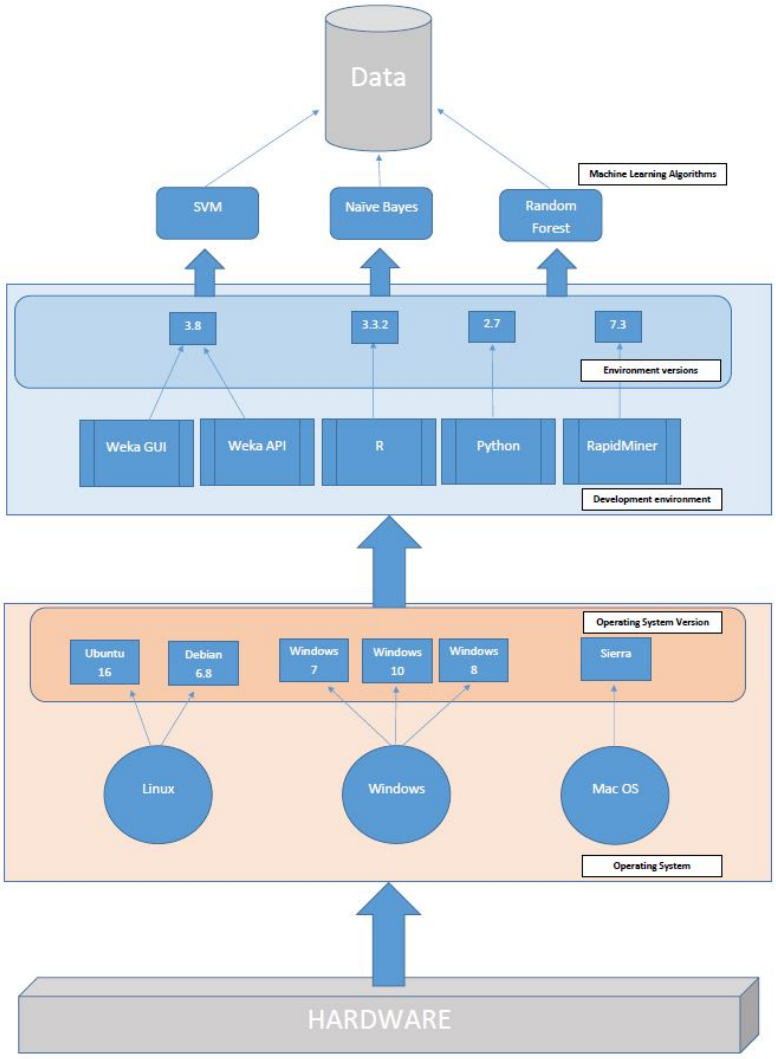

Figure 3.4: The second Experimental scheme is presented in this Figure. Each experiment used the same Hardware structure, all of the three machine learning models and the same database. Each model was implemented in different environments (Weka API 3.8, Weka GUI 3.8, Python 2.7, R 3.3.2 and RapidMiner 7.3) run on six different operating systems - Windows 10, Windows 8, Windows 7, Max OS Sierra, Linux Ubuntu 16 and Linux Debian 6.8.

For the purposes of the experiments described above, a Vmware Workstation Pro 12.5

[24] was installed as a virtual station on a Windows 10, 64-bit computer. Each of the operating systems and their respective versions were installed inside a virtual machine using licensed virtual machine ISO files.

In order for environments based on a graphical interface such as RapidMiner, WEKA GUI and R to be able to run on every operating system we needed to download and install the proper software executable file from their respective websites, install the software, transfer the the "csv" data file and either transfer the script (in the case of R) or implement the models as shown in Appendix A and D (RapidMiner and Weka GUI).

For each of the operating systems to be able to run the same Python 2.7 script, the following steps were necessary:

1. Installing Python [25]
2. Adding Python to Path
3. Installing package manager Python package index (PIP)[26]
4. Installing dependencies needed for the Python implementation
5. Transfer the code together with the "csv" file to the OS.
6. Running the script in Python via terminal

Running the WEKA API on each operating system requires:

1. Downloading the Eclipse IDE [27]
2. Import weka exe package through jar files from the buil path function in Eclipse.
3. Transfer the code together with the "csv" file to the OS.
4. Running the script via Eclipse terminal.

As a result of the many differences between the operating systems, tracking the progress and the output predictions from the models run in Python is done by using Log functions in the Python script. The log functions have the purpose of printing out a status after every important method in the implementation. This have helped us follow the experimentation flow and the outputs that are given in some mid processes. The analytical methods used in this thesis for comparing and gathering results from all experiments are discussed in the next section.

[24] http://www.vmware.com/products/workstation/workstation-evaluation.html
[25] https://www.python.org/downloads/
[26] https://pypi.python.org/pypi/pip
[27] https://eclipse.org/downloads/

Tracking the results of the others environments, is done by their respective console application (R and Weka API) or by manually checking the processes (Weka GUI and RapidMiner)

3.6 Analysis methods

To be able to analyze and compare the results from all of the experiments from the two experimental schemes we create two different analyzing methods. The first one is based on analyzing the overall model predictions. For each algorithm implemented into different environment and run on different operating systems we are able to collect and compare the following measures:

- Model **accuracy** represents the number of correct predictions from the model divided by the number of total predictions made.
- Model **classification error** represents the number of incorrect classified instances divided by the number of total predictions.
- **Cohen's Kappa value** represents a statistical score that defines the level of agreement between two annotators on a classification problem.
- **Mean squared error** represents a risk function measuring the average squares of the errors or deviation.

For a better representation of the results a confusion matrix is used explaining the prediction results for each of the models.

The second analytical model that we use, is based on a deeper understanding of cross validation function in each of the implementations. The fact that many of the implementations used in this thesis such as RapidMiner and Weka don't provide their algorithms source code and have a poor scientific documentation, we are able to extract and analyze only the variance and accuracy for each of the folds generated. Using this, we are able to analyze and compare the statistical significance difference on the accuracy and variance between each experiment.

To be able to compare the results produced by Python 2.7 environment from different operating systems, we create a log file in Python as shown in Figure 3.5 with the following structure:

1. The list of all of the audio files processed from the csv file. The list represents the order of the audio item as they are processed by the OS.
2. The name of different classes assigned from the feature extractor.
3. The name of the machine learning model that was used in that run together with its parameters.

4. The list of the IDs from the training set and test set for each of the Folds.
5. The variance for each test set and training set for each fold in the 10 cross validation.
6. A confusion matrix.
7. As mentioned above, the following image represents the statistical measures such as: Accuracy, Recall, Mean Squared Error, classification error, Kappa value.

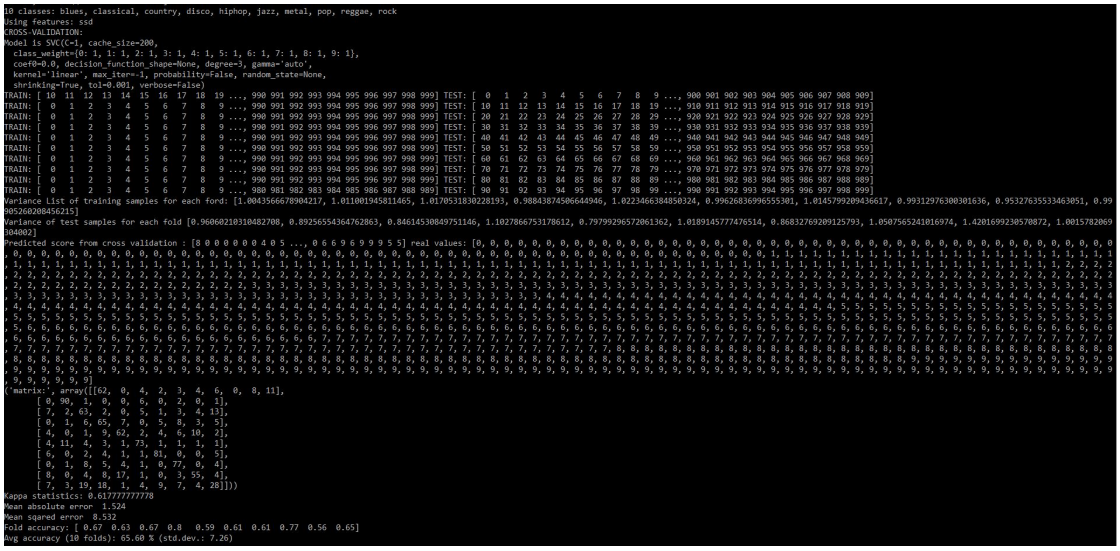

Figure 3.5: Example of the output of the Log file in Python

3.6.1 McNemar's test

To be able to compare different development environments results, for each one of the machine learning algorithms we create a McNemar's test. The McNemar's test is a statistical test used on paired nominal data. Even if it is mostly used to compare the performance of two different machine learning algorithms on the same dataset, we used it to compare a single machine learning model which was implemented on two different development environments. In order to apply this test, a 2 X 2 contingency table is created which tabulates the outcome of two tests from two different implementations. The 4 values that this test relies on are:

1. The number of instances misclassified by both implementations (c_{00}).
2. The number of instances misclassified by the first implementation f_1 but correctly classified by the second implementation f_2 (c_{01}).
3. The number of instances misclassified by the second implementation f_2 but correctly classified by the first implementation f_1 (c_{10}).

4. The number of instances correctly classified by both implementations (c_{11}).

The McNemar χ^2 statistic is given by the following formula:

$$\chi^2_{MC} = \frac{(|c_{01} - c_{10}| - 1)^2}{c_{01} + c_{10}} \tag{3.3}$$

If $c_{01} + c_{10} \geq 20$ then χ^2_{MC} is compared to the χ^2 statistics. If χ^2_{MC} exceeds the $\chi^2_{1,1-\alpha}$ statistics, then we can reject the null hypothesis that assumes that f_1 and f_2 perform equally with 1-α confidence.

While performing the McNeuman test we take a probability level alpha of 0.05 to be able to determine if differences between two implementations of a model are statistically significantly different .

3.6.2 Two Sample Kolmogorov-Smirnov Test

After the analysis of the first experimental scheme we came to a conclusion that different development environments produce different test and training sets in each fold of the cross validation method. In oder to show whether the distributions behind the individual 10 folds are statistically significantly different we used the Two Sample Kalmogorov-Smirnov Test which is mainly used to test whether two samples come from the same distribution. In the following paragraphs we will explain the test more in details.

The Kolmogorov-Simirnov two sample test statistic is used to test whether two data samples come from the same distribution It is important to note that the common distribution that the test is referring too is not well specified in the literature. The test is mostly based on the empirical distribution function (ECDF) which given N data points $Y_1, Y_2, ...Y_N$ is defined as

$$E_N = \frac{n_i}{N} \tag{3.4}$$

where n_i is the number of points less than Y_i. This function represents a step function which increases by $\frac{1}{N}$ at the value of each data point. The null hypothesis that the Kolomogorov-Simirnov two sample test statistic is based upon is states that: The two examples come from a common distribution. The test is defined as

$$D = |E_1(i) - E_2(i)| \tag{3.5}$$

where E_1 and E_2 present empirical distribution functions. The hypothesis regarding the distribution form is rejected if the test statistic D, is greater than the critical value obtained from a table. In order to reject or acept the hypothesis, we used the p-value calculated using the "Real Statistic" package using Excel [28]. While performing the Kolomogorov-Simirnov two sample test statistic we take a probability level alpha of 0.05 to be able to determine whether the differences in distributions behind the individual 10 folds from two development environments are statistically significantly different.

[28] http://www.real-statistics.com/non-parametric-tests/two-sample-kolmogorov-smirnov-test/

3.7 Summary

This chapter was dedicated on explaining the methodology and structure used in creating the experiments in this master thesis. We defined two different experimental schemes which were used to analyze the influence of different environments and operating systems on classification machine learning algorithms and their results. As we previously discussed, the first experimental scheme was using different development environments such as Weka GUI, Weka API, Python, R and RapidMiner. All of them had their own different interface and stick parameter setting for each of the algorithms that were analyzed. We observed the problem of finding structured and relevant documentation, providing detail explanation and source code for machine learning packages used by some of the environments. That is why certain parameters which were available in all of the environments were set the same.

The second experimental scheme focused on running a Python 2.7, RapidMiner, R, Weka 3.8 API and Weka 3.8 GUI programs on different operating systems and their different versions.Because of the operational differences that the operating systems have, evaluating the results from the experimental scheme is done by creating a Log file using printout functions that explain and regulate the processes in the Python script. Via the Log file we are able to analyze the different results and extract inside informations that will help feature researchers when they perform reproducibility on machine learning algorithms using different operating systems.

In the last part of this chapter we discussed the statistical parameters that were extracted from each experiment. The machine learning algorithms are mostly evaluated by their performance accuracy together with some other statistical measures such as kappa value, recall, mean squared error etc. In order to analyze the statistical significant difference between algorithms implemented in different environments, a McNeuman test was used.

In the next chapter we will go deeper into explaining the results for each of the experiments in both experimental schemes separately. Throughout the next chapter we will see the difficulties and the sensitivity of using different operating systems and environments.We will also analyze the flexibility and the sensitivity of the Support Vector machine, Naive Bayes and Random Forest algorithms.

CHAPTER 4

Results

In this chapter the results from all of the experiments done in the two experimental schemes are presented. First, we start by showing the analysis and arguments behind the first experimental scheme, dedicated in exploring the influence of changing the development environment when trying to reproduce machine learning algorithms. Analysis of machine learning performance measures using different development environments such as Weka GUI, Weka API, Python,R and RapidMiner and their effect on machine learning performance is discussed in details.

In the second section of this chapter, the second experimental scheme is approached. Through the results of the analysis from machine learning algorithm performances on different operating systems, the flexibility of reproducing a machine learning algorithm is observed. Furthermore, the influence of different operating system versions on a chosen model are discussed.

At the end we will give a short overview of the results and and briefly introduce the aspects of the topic discussed in the following chapter.

4.1 Variations across algorithm Implementations

In this section, the results of the first experimental scheme will be analyzed and discussed. The goal of these experiments was not to compare the performance results between the algorithms themselves, but to compare predefined performance measures for a machine learning model in different environments. In order to do that, from each of the experiments we extract the model's accuracy, classification error, kappa value and recall. Using this measures we have a good overview of how a machine learning algorithm performs in each experiment.

In order to compare the difference between each development environment performances, a McNemar's test was performed exploring the statistical significant difference between

implementations of a model. Also we had a closer look and analyzed the variance from each testing data in each fold created by every environment and used the Two Sample Kolmogorov-Smirnov Test to test whether folds created in two different environments come from the same distribution.

4.1.1 Evaluation criterion

By adopting the McNemar's test to evaluate the performance of a classification algorithm implemented in different environments, the following criterion is defined: An algorithm developed under a specific environment is regarded as successful if it can predict the class of an instance correctly. On the other hand, it is regarded as "failed" when it predicts an incorrect label class for an instance.

Using this criteria, the χ^2 scores are calculated using McNemar's test for each of the machine learning algorithms developed under 5 different environments. As discussed before, each of the algorithms implementations had the same parameters, as parameters tunning may favor one implementation of the same algorithm to produce different results.

The null hypothesis(H0) for the experimental design suggests that two implementations of the same model give statistically significantly different results whereas the alternative hypothesis (H1) claims otherwise, suggesting that the results are statistically not significantly different as shown in Equation 4.1 where C_1 represents the first implementation of the model and C_2 represents the second implementation of the model.

$$\begin{aligned} H0 &: C_1 = C_2 \\ H1 &: C_1 \neq C_2 \end{aligned} \tag{4.1}$$

Under the null hypothesis, the χ^2 has a chi-squared distribution with 1 degree of freedom. If the value is bigger than 3.84 the chi-squared distribution with 1 degree of freedom gives p-value smaller than 0.05 which require us to reject the null hypothesis.In other words if the p-value has a value below probability level-alpha of 0.05, this provides sufficient evidence to reject the null hypothesis. If the p-value is above 0.05 we don't reject the null hypothesis and conclude that the accuracy of both implementations of the same model are statistically significantly different. In order to calculate the χ^2 scores, the classification results from all of the models developed in different environments must be identified for each individual instance.

In order to perform this type of testing, output results from all different environments were necessary. Due to the fact that the cross validation algorithm creates different folds in each environment (choosing test and training folds in each environment is different),to be able to compare the prediction of the same instances from two different implementations we needed to find out which instances were tested in each fold of the cross validation. In Python 2.7 we created a script which extracted the ID of each instance and the respective predicted class label in a "csv" file. In R, the package caret provides a data frame including each test instance ID, prediction output label and kappa value. Weka GUI shows the

incorrect classified instances via the "output predictions" options of the classifier which displays a "+" in the output next to the instance which has been incorrectly classified. To be able to match every instance ID from the classifier we extract a "csv" file from the knowledgeflow segment of Weka. The Weka API and RapidMiner have an internal buffer inside the cross validation function which provides the IDs and the output values for each test instance.

The results from every implementation using different models were stored into an Excel table where miss-classified instances were assigned to 0 and correctly classified instances to 1. Using the advantages of Excel, a pivot table was created representing 2×2 contingency table for each pair of implementations. For each contingency table, a χ^2 was calculated.

4.1.2 Results from changing the development environment in reproducibility of machine learning algorithms.

We will start analyzing the results gathered from implementing the Support Vector Machine algorithm in Python, Weka GUI, RapidMiner, R and Weka API. When trying to reproduce a research based on machine learning algorithms, the authors not always state the version of the development environment under which the research was created. Due to the fact that platforms update and change their versions drastically nowadays, we also created an experiment analyzing the output accuracy of machine learning algorithms developed in different versions of Weka GUI (3.6 & 3.7 & 3.8), Weka API (3.6 & 3.7 & 3.8) and Python (2.7 & 3.6). Each of of the development environments were run on Windows 10 operating system on the same Hardware.In order to be able to compare their performances,for each one of the implementations, we calculated the accuracy, the classification error, Kappa value and Recall values. In Table 4.1 the results gathered from running the Support Vector Machines algorithm in all of the environments and their different versions is presented.

As shown on the Table 4.1 changing the version of the same environment doesn't change the prediction result of Support Vector Machine algorithm in our experiments. On the other hand,the calculated measures taking into an account different development implementation showed different results. Python 2.7 calculated accuracy of 65.60%, approximately 9 % lower than the rest of the implementations. RapidMiner had the best accuracy of 72.40%. Surprisingly, Weka API performed with 71.80% accuracy and Weka GUI with 70.20 % accuracy even though they represent the same platform and their documentation states that they use the same Java code.

This results show that implementing the same algorithm and testing its performance with 10 fold cross validation on different platforms give different results. To be more specific we implemented the McNemar's test discussed above for each pair of implementations. The Table 4.2 shows the results of the test providing the χ^2 value for each test. If the value is bigger than 3.84 the chi-squared distribution with 1 degree of freedom gives p-value smaller than 0.05 which require us to reject the null hypothesis.

Enviorments	Accuracy	Classification error	Kappa	Recall
Python 2.7	65.60%	34.40%	0.618	0.656
Python 3.6	65.60%	34.40%	0.656	0.8532
Weka 3.6	70.20%	29.80%	0.6689	0.7020
Weka 3.7	70.20%	29.80%	0.669	0.7020
Weka 3.8	70.20%	29.80%	0.6689	0.7020
Rapid Miner	72.40%	27.60%	0.693	0.704
R	70.90%	29.10%	0.6701	0.709
Weka 3.6 API	71.80%	28.30%	0.6856	0.7180
Weka 3.7 API	71.80%	28.30%	0.6856	0.7180
Weka 3.8 API	71.80%	28.30%	0.6856	0.7180

Table 4.1: Results from running Support Vector Machine algorithm using cross validation in 5 different development environments such as: Python (2.7 & 3.6), Weka GUI (3.6 & 3.7 & 3.8), RapidMiner 7.2, R 3.3.2 and Weka API (3.6 & 3.7 & 3.8) run on Windows 10 operating system

By looking at the Mc Nemar's test results for the Support Vector Machine algorithm run on different environment (Table 4.2), one can deduce that SVM implemented in Python provides results statistically significantly different from the rest of the implementations. The p-values calculated in this cases were < 0.05 which allowed to reject the null hypothesis. The χ^2 value calculated for the Mc Nemar's test done between RapidMiner and Weka GUI was 3.205 which has a p-value between 0.10 and 0.05. All of the implementation pairs between Weka API, RapidMiner,R and Weka GUI are not statistically significantly different than each other. In other words, taking the null hypothesis which states that there is a difference between two implementations of a SVM model is not statistically significantly different between those implementations.

For SVM	RapidMiner	R	Python	Weka API
Weka GUI	3.205	0.23	**12.5**	1.25
RapidMiner		3.66	**31.79**	0.094
R			**6.0575**	3.3
Python				**19.1804**

Table 4.2: Mc Nemar's Test Results for Support Vector Machine algorithm

Naive Bayes as the second type of classification algorithm was implemented in all of the above discussed platforms. Because of its simplicity, the results described in Table 4.3 were unexpected. The Naive Bayes algorithm run on the same dataset with the same set of parameters in each implementation. Due to the fact that not all of the environments provide specific documentation of their algorithms implementation, we

considered the hypothesis that every implementation of Naive Bayes was done the same. Each implementation was given the same random number generator of 1 for the cross validation algorithm.

Enviorments	Accuracy	Classification error	Kappa	Recall
Python 2.7	45.30%	54.70%	0.3922	0.45
Python 3.8	45.30%	54.70%	0.3922	0.45
Weka GUI 3.6	50.00%	50.00%	0.444	0.50
Weka GUI 3.7	50.00%	50.00%	0.444	0.50
Weka GUI 3.8	50.00%	50.00%	0.444	0.50
Rapid Miner	48.90%	51.10%	0.432	0.48
R	52.00%	48.00%	0.466	0.52
Weka API 3.6	49.80%	50.20%	0.4422	0.498
Weka API 3.7	49.80%	50.20%	0.4422	0.498
Weka API 3.8	49.80%	50.20%	0.4422	0.498

Table 4.3: Results from running Naive Bayes algorithm using cross validation in 5 different development environments such as: Python(2.7 & 3.6), Weka GUI(3.6 & 3.7 & 3.8), RapidMiner 7.2, R 3.3.2 and Weka API(3.6 & 3.7 & 3.8) run on Windows 10 operating system

As the table 4.3 shows,changing the version of the environments tested in this experimental scheme doesn't present different results when it comes to implementing Naive Bayes algorithm. Further, the experiments show that the accuracy of Naive Bayes under different environments is different. As in the Support Vector Machine experiment, Python obtained the lowest accuracy. The accuracy produced by Weka API and Weka GUI differ in 0.20%, which again confirms the hypotheses that Weka returns different results under different development platform. Niave Bayes implemented in R returns the highest accuracy of 52.00%.

Table 4.4 shows the X^2 calculated by the McNemar's test which shows the statistical significant difference between a pair of implementations running Naive Bayes. The X^2 calculated for RapidMiner and R, Python and RapidMiner, R and Python, Weka API and R and Python and Weka API gave results with p- values smaller than 0.05. In this cases the null hypothesis stated before was rejected and we can say that the these pairs of implementations are statistically significantly different. On the other hand Weka GUI and Weka API have a p-value between 0.10 and 0.05 which does not allow us to reject the null hypothesis. In Table 4.4 all of the number which are assigned in bold represents pairs of implementations for which the McNeumar's test proves that their are statistically significantly different. The pairs of implementations that Weka GUI has with RapidMiner, R, Python and Weka API together with pairs such as RapidMiner and Weka API using McNeumar's test show that, regardless of the fact that the implementation accuracies are similar,the Mc Neumar test show that they are not statistically significantly different.

For NB	RapidMiner	R	Python	Weka API
Weka GUI	0.086	0.93	2.332	0.001
RapidMiner		**8.346**	**14.56**	0.816
R			**33.885**	**5.8**
Python				**25.142**

Table 4.4: Mc Nemar's Test Results for Naive Bayes algorithm

Random Forest as an ensemble learning algorithm was the hardest to implement and analyze. Its random factor provided different kind of trees in each implementation, even if we assigned the same random seed number 15325 in each experiment. The results of the accuracy calculated for each implementation of Random Forest are shown in the Table 4.5.

Enviorments	Accuracy	Classification error	Kappa	Recall
Python 2.7	52.90%	47.100%	0.4766	0.53
Python 3.6	52.90%	47.100%	0.47666	0.53
Weka 3.6 GUI	62.00%	37.20%	0.5866	0.62
Weka 3.7 GUI	61.20%	38.80%	0.5689	0.61
Weka 3.8 GUI	61.20%	39.90%	0.5567	0.61
Rapid Miner	39.40%	60.600%	0.3270	0.39
R	61.00%	39.00%	0.5666	0.39
Weka 3.6 API	62.40%	37.60%	0.58	0.624
Weka 3.7 API	62.10%	38.00%	0.577	0.621
Weka 3.8 API	62.10%	37.90%	0.5789	0.621

Table 4.5: Results from running Random Forest algorithm using cross validation in 5 different development environments such as: Python(2.7 & 3.6), Weka GUI(3.6 & 3.7 & 3.8), RapidMiner 7.2, R 3.3.2 and Weka API(3.6 & 3.7 & 3.8) run on Windows 10 operating system

Changing the version of Python implementation doesn't give us different accuracy results as shown in Table 4.5. On the other hand we can elaborate that the newest version of Weka GUI and API gives us different accuracy results when running a Random Forest model. Changing the version from 3.6 to 3.7 or 3.8 the results decreased their accuracy for 0.30% in Weka API and 0.80% in Weka GUI We can argue that the lack of detailed documentation explaining the implemented source code of the algorithms makes it very hard to analyze the reason why different versions of Weka provide different results.By creating their versions Weka is adding new hyper-variables that can be manually changed when implementing Random Forest algorithm. We can only assume that the change of

the accuracy in different versions is a result of this updates.

Furthermore, analyzing the results produced by running Random Forest in different environments, we can argue that each of the development environments present different results. The biggest difference between results is calculate in RapidMiner environment with an accuracy approximately 21% smaller than all the others. Weka API and Weka GUI differ a total of 0.90% from each other. Python 2.7 gives 52.90% accuracy which is 9.10 % smaller than the highest accuracy provided by Weka API. Even if the difference in performance is very big in each environment, the McNemar's test was calculated for each pair of environments The results of the test are shown in Table 4.6.

For RF	RapidMiner	R	Python	Weka API
Weka GUI	0.248	**57.05**	**22.858**	**64.91**
RapidMiner		**137.574**	**52.967**	**142.502**
R			**27.7056**	0.77
Python				**34.65**

Table 4.6: Mc Nemar's Test Results for Random Forest algorithm implementations

Random Forest implemented in Weka API and R and Rapid Miner and Weka GUI are the only two pairs of experiments which are not statistically significant different. Regardless of the fact that Weka API and Weka GUI have similar accuracy results, they still have a p-value smaller than 0.05 which allows to reject the null hypothesis and make them statistically significantly different.

From all of the experiment results shown above we can conclude that running a 10-fold cross validation of machine learning algorithms using different environments outputs most of the time statistically significantly different results(17 out of 30 pairs are statistically significantly different).

With an aim to go deeper into investigating the reason why all of the algorithms present different results on different development environments and test the hypothesis that all of the different result are based on different random split of the folds in the cross validation in each environment, we tested all the experiments from the first experimental scheme using a fixed one time 10-fold cross validation evaluation. We took the fold that was produced by the R implementation which included folds with variances and created new scripts for each environment testing the Accuracy,Classification Error,Kappa and the Mean Squared Error for all of the machine learning models.

By running Support Vector Machines and Naive Bayes algorithms on the different development environments using a fixed one time 10 fold cross validation we achieved the same results for each experiments. On the other hand we realized that the implementation of Random Forest in this experiments gave different results. We argue that this is certainly because of the randomness factor that Random Forest has in its nature. The accuracies

of this experiments for each of the algorithms run on all of the environments are shown in Table 4.7.

Model / Environment	Weka GUI	RapidMiner	R	Python	Weka API
Support Vector Machine	70.90%	70.90%	70.90%	70.90%	70.90%
Naive Bayes	52.00%	52.00%	52.00%	52.00%	52.00%
Random Forest	60.00%	61.70%	61.00%	61.25%	61.01%

Table 4.7: Accuracy of SVM, Naive Bayes and Random Forest implemented in all 5 different environments using one time fixed 10-folds cross validation

In order to prove that the development environments also calculate the same prediction results for all of the models using training/testing set we tested the same experiments using specifically created test and training set from the database. The test and the training set were done manually by splitting the database into 90:10 training and test set. With this experiment we wanted to explore the influence of the 10-fold cross validation in the results.

Model / Environment	Weka GUI	RapidMiner	R	Python	Weka API
Support Vector Machine	65.00%	65.00%	65.00%	65.00%	65.00%
Naive Bayes	38.00%	38.00%	38.00%	38.00%	38.00%
Random Forest	43.00%	33.00%	49.00%	46.00%	43.00%

Table 4.8: Accuracy of SVM, Naive Bayes and Random Forest implemented in all 5 different environments using a single test and a training set split

As we can see from Table 4.8, when we run the same algorithms on the same implementations with assigned test and training set we got the same accuracy results from each of them.On the other hand, implementing Random Forest as in the previous experiment gives different results in each environment, regardless on the dataset evaluation method.

From both of the experiments explained above we can confirm the hypothesis that each environment produces different random splits from the dataset in order to perform a cross validation evaluation. Each of the environments selected different training and test instances in the 10-cross validation evaluation and different random instances in creating the trees in Random forest.

In order to understand the random training and testing folds created from each development environment we had a closer look on the variance of the test set for each fold produced by each environment. The Table 4.9 shows that each of the environment creates a test folds with an average of variance of accuracy in range od 0.87 to 1.5.

Folds/Environments	Python 2.7	Weka 3.7GUI	RapidMiner	R	Weka 3.7 API
Fold 1	0.961	0.908	0.934	1.122	0.851
Fold 2	0.892	0.925	1.237	0.800	0.915
Fold 3	0.846	0.899	0.860	1.025	1.002
Fold 4	1.108	0.963	0.892	0.905	0.999
Fold 5	0.798	1.434	1.0218	1.147	0.999
Fold 6	1.018	0.859	1.0136	1.092	1.096
Fold 7	0.868	1.121	0.935	0.871	1.081
Fold 8	1.050	0.959	0.950	1.096	1.256
Fold 9	1.420	1.081	0.998	0.978	0.852
Fold 10	1.001	0.887	1.19	0.957	0.959

Table 4.9: Variances across the test set from each test fold created in each environment

By performing a Kolmogorov Smirnov two sample test as explained in the Methodology section we tested for each of the environments whether the difference in distributions behind the individual folds are statistically significantly different. We tested the following null hypothesis H_0: both samples come from a population with the same distribution. The p-value which determines whether we reject the null hypothesis (at significance α=0.05) for each of the tests is presented in the Table 4.10.

	Weka GUI	RapidMiner	Weka API	R
Python	**0.022016528**	0.301869753	**0.045035941**	0.199508675
Weka GUI		0.305953592	0.269938689	0.797017656
RapidMiner			0.643351056	0.260668031
Weka API				0.827855188

Table 4.10: Kolmogorov Smirnov two sample test p-value testing whether distributions behind the individual folds are statistically significantly different

From the Kolmogorov Smirnov two sample test taking into the consideration the p-value as shown in the Table above, we can conclude that there is a statistical significant difference between the difference in distribution for each of the folds created by the Python implementation and the Weka GUI/API implementation. All of the others pairs of tested implementations presented a p-value larger than 0.05 showing that there is not a statistical significant difference between the differences in the distribution of the folds created.

In the next section, the influence of different operating systems running machine learning algorithms when trying to reproduce a research will be analyzed. Having in mind the fast and continuous change in operating system versions, results from different versions

of Windows and Linux will be discussed.

4.2 Variations across Operating System Versions

In this section the results of executing the second experimental scheme explained in Figure 3.4 are shown. Using Vmware Workstation Pro 12.5 virtual machine, virtual images of operating systems such as Linux Ubuntu 16, Linux Debian 6.8, Windows 8, Windows 7 and Mac Sierra were installed on a base operating system Windows 10 (64 bit). On each of the operating systems, all of the development environments mentioned before were installed (Python 2.7, RapidMiner, R, Weka 3.8 API, Weka 3.8 GUI) in order to analyze the influence of the operating system using each environment in reproducibility of machine learning results.

Each environment was successfully installed in all of the operating system, where a specifically defined script or design process flow was run. In every environment we implemented Support Vector Machines,Naive Bayes and Random Forest,each using the "csv" file containing features from an audio GTZAN collection as an input. Furthermore, a model was created from each one of the machine learning algorithms. The evaluation of the model was done using one time 10-fold cross validation with a random seed of 1.

In order to compare the results of the models run in different operating systems the accuracy for each model were calculated. The accuracy of the models from using R, RapidMiner, Weka API, Weka GUI and Python implementations run in different operating systems are shown in Tables 4.11,4.12, 4.13, 4.14 4.15 respectively.

Operating System / Model	Support Vector Machine	Naive Bayes	Random Forest
Windows 10	70.90%	52.00%	61.00%
Windows 8	70.90%	52.00%	61.00%
Windows 7	70.90%	52.00%	61.00%
Linux Ubuntu 16	70.90%	52.00%	61.00%
Linux Debian 6.8	70.90%	52.00%	61.00%
Mac OS Sierra	70.90%	52.00%	61.00%

Table 4.11: Accuracy from Support Vector Machine, Naive Bayes and Random Forest models implemented in R run on different operating systems

As shown in the Tables 4.12,4.13,4.14 and 4.15, we can conclude that all of the performance accuracies stay the same when we implement the models in Python,R,RapidMiner,Weka API,Weka GUI and run them on each of the operating systems separately. This is the case when using a specific "csv" file as an input and the same code or structure for the environments The random forest shows the same results here, because of the fact that the random generator for each implementation doesn't show any changes while run on different operating systems. In the next subsection, we will go deeper into trying the

Operating System / Model	Support Vector Machine	Naive Bayes	Random Forest
Windows 10	72.40%	48.90%	39.40%
Windows 8	72.40%	48.90%	39.40%
Windows 7	72.40%	48.90%	39.40%
Linux Ubuntu 16	72.40%	48.90%	39.40%
Linux Debian 6.8	72.40%	48.90%	39.40%
Mac OS Sierra	72.40%	48.90%	39.40%

Table 4.12: Accuracy from Support Vector Machine, Naive Bayes and Random Forest models implemented in RapidMiner run on different operating systems

Operating System / Model	Support Vector Machine	Naive Bayes	Random Forest
Windows 10	71.70%	49.80%	62.00%
Windows 8	71.70%	49.80%	62.00%
Windows 7	71.70%	49.80%	62.00%
Linux Ubuntu 16	71.70%	49.80%	62.00%
Linux Debian 6.8	71.70%	49.80%	62.00%
Mac OS Sierra	71.70%	49.80%	62.00%

Table 4.13: Accuracy from Support Vector Machine, Naive Bayes and Random Forest models implemented in Weka 3.7 API using Java 8 run on different operating systems

Operating System / Model	Support Vector Machine	Naive Bayes	Random Forest
Windows 10	70.20%	50.00%	61.20%
Windows 8	70.20%	50.00%	61.20%
Windows 7	70.20%	50.00%	61.20%
Linux Ubuntu 16	70.20%	50.00%	61.20%
Linux Debian 6.8	70.20%	50.00%	61.20%
Mac OS Sierra	70.20%	50.00%	61.20%

Table 4.14: Accuracy from Support Vector Machine, Naive Bayes and Random Forest models,implemented in Weka 3.7 GUI run on different operating systems

Operating System / Model	Support Vector Machine	Naive Bayes	Random Forest
Windows 10	65.60%	45.30%	52.90%
Windows 8	65.60%	45.30%	52.90%
Windows 7	65.60%	45.30%	52.90%
Linux Ubuntu 16	65.60%	45.30%	52.90%
Linux Debian 6.8	65.60%	45.30%	52.90%
Mac OS Sierra	65.60%	45.30%	52.90%

Table 4.15: Accuracy from Support Vector Machine, Naive Bayes and Random Forest models implemented in Python 2.7 run on different operating systems

reproduce the Python code provided from the study "Evaluation of Feature Extractors and Psycho-acoustic Transformations for Music Genre Classification" [LR05] in order to explore the influence of operating system when reproducing a study which reads files from a file directory.

4.2.1 Results calculated from reproducing the Python code from "Evaluation of Feature Extractions and Psycho-acoustic Transformations for Music Genre Classification" study using different operating systems

In order to take this research further, we tried to reproduce the study of Lily and Rauber [LR05] using the code that they provided on their website[1]. In their proposition they are taking the GTZAN database, train it and then predict the accuracy of the models using one time 10 folds cross validation. Each audio file is read from the file directory and an algorithms proposed in this study is implemented which extracts the 168 attributes using the Psycho-acoustic Transformation After transforming all of the audio files a "csv" file is created on which a model is applied and an accuracy is predicted using a one time 10 fold cross validation. The difference between the secound experimental scheme analyzed above and this experiment was the usage of "csv" file as an input data vs using audio files as an input data. The feature extraction part of the process performs identically, i.e. that it extracts the same numeric features for each of the audio files. (as, otherwise, differences in he extraction process might explain the difference in the upcoming results).

The issue that we came across while trying to reproduce this study by changing the operating system only, was the un-sustainable prediction accuracy that we calculated from each experiment. This particular set of experiments showed different results run in every operating system. As shown in the Table 4.16 the Support Vector Machine model calculates an accuracy of 65.90% run in all of the versions of Windows operating systems and Mac OS Sierra operating system. On the other hand Linux Ubuntu 16 and Linux Debian 5.8 calculate different accuracy of 69.70% and 68.90%. The accuracy for Naive

[1] http://ifs.tuwien.ac.at/mir/audiofeatureextraction.html

Bayes have a similar behavior. Random Forest run on all of the operating systems gives a different accuracy result for each of them.

Operating System / Model	Support Vector Machine accuracy./std.dev.	Naive Bayes accuracy./std.dev.	Random Forest accuracy./std.dev.
Windows 10	65.90% / 6.01	45.40% / 5.82	54.80% / 7.93
Windows 8	65.90% / 6.01	45.40% / 5.82	56.20% / 5.93
Windows 7	65.90% / 6.01	45.40%/ 5.82	53.70% / 4.22
Linux Ubuntu 16	69.70% / 5.00	49.90% / 3.67	60.90% / 5.36
Linux Debian 6.8	68.90% / 4.98	49.60% / 4.21	59.90% / 4.56
Mac OS Sierra	65.90% / 6.01	45.40% / 5.82	54.30% / 5.44

Table 4.16: Accuracy and Standard Deviation from Support Vector Machine, Naive Bayes and Random Forest models implemented in Python 2.7 run on different operating systems while trying the reproduce the Python code from "Evaluation of Feature Extractors and Psycho-acoustic Transformations for Music Genre Classification" [LR05]study.

The average standard deviation calculated from all of the folds is shown in Table 4.16. Both of the versions of Linux give different standard deviation than Windows 10, Windows 7, Windows 8 and Mac OS .

The difference between all operating systems made very difficult to compare all of the system behaviors while running the algorithms. To be able to track the processes in each operating system a log output is created in Python as described in the Methodology chapter of this thesis (Figure 3.5).

Due to the fact that all of the algorithms results differ drastically while running on Windows and Linux we analyzed the log files. As shown in Figure 4.1, Windows read the files in alphabetic order. On the other hand Linux read and list directory files in random order as shown in Figure 4.2

By backtracking the Python code we realized that the command *os.walk*, which is used to read the files from the directory, generates the file names in a directory tree by walking either top down or bottom up. For each directory in the tree created it yields a tuple for each file consisting of directory path, directory names, file name. Because of the fact that Windows and Linux have different filing systems, the order of the files read in both experiments was different. This influenced the creation of cross validation folds which affected the performance accuracy.

We repeated all the experiments by sorting the audio files after they have being read and a providing a static random seed for the Random Forest algorithm. The results were the same as in the experiment we run the Python scipt using an "csv" file as in input shown in Table 4.16. This confirms the hypothesis that by implementing forced sorting and static random seed for Random Forest the operating system doesn't change the prediction accuracy of machine learning algorithms implemented in Python. Also,

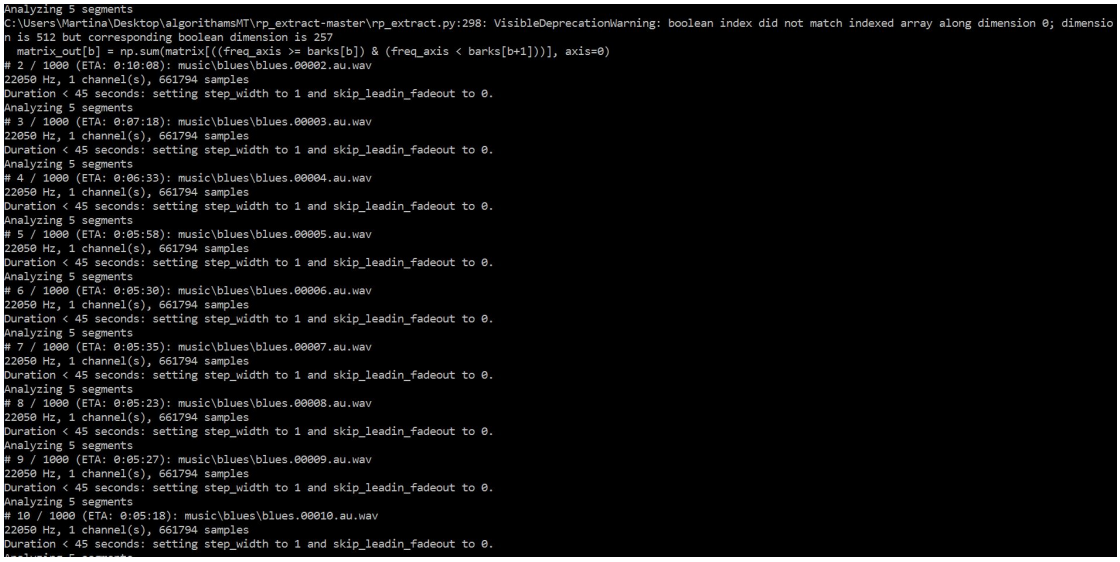

Figure 4.1: Screen shot of a log file output for running Support Vector Machine on Windows 10

we can conclude that different versions of operating systems don't have any influence in the outcome. Meanwhile the researchers need to be very careful when implementing a Python script which uses an operating system command because of the operating systems differences in file processing.

4.3 Summary

In this chapter all the results from each experimental scheme were discussed and analyzed in detail.The summary table from all of the experiments made is shown in Table 4.18. We analyzed the experiments from the first experimental scheme by comparing the accuracy results for each implementation for Support Vector Machine, Naive Bayes and Random Forest algorithms. To be able to define the statistical significant difference between the results we calculated the McNeuman's test for each pair of experiments based on a specific machine learning algorithm. The results showed that some algorithms such as Random Forest calculate different accuracy and statistically not significant different results when reproduced on different environments. Furthermore, we analyzed the influence that cross validation has on this changes, by creating additional experiments using specifically defined training and test set and a fixed one time 10-fold cross validation technique The results shows that the accuracy doesn't change when a specific folds or training/test set is given for evaluation. On the other hand, cross validation generates different folds across implementations with distributions behind the individual 10 folds not statistically significantly different.

Also included in the first section of this chapter we discussed the results of experiment

```
martinanaumovska@ubuntu:~/Documents/MT/rp_extract-master$ python rp_classify.py
-cv music
Performing feature extraction from  music
# 1 / 1000 (ETA: --:--:--): music/metal/metal.00082.au.wav
22050 Hz, 1 channel(s), 661504 samples
Duration < 45 seconds: setting step_width to 1 and skip_leadin_fadeout to 0.
Analyzing 5 segments
# 2 / 1000 (ETA: 0:11:53): music/metal/metal.00015.au.wav
22050 Hz, 1 channel(s), 661504 samples
Duration < 45 seconds: setting step_width to 1 and skip_leadin_fadeout to 0.
Analyzing 5 segments
# 3 / 1000 (ETA: 0:13:01): music/metal/metal.00043.au.wav
22050 Hz, 1 channel(s), 661794 samples
Duration < 45 seconds: setting step_width to 1 and skip_leadin_fadeout to 0.
Analyzing 5 segments
# 4 / 1000 (ETA: 0:13:46): music/metal/metal.00045.au.wav
22050 Hz, 1 channel(s), 661794 samples
Duration < 45 seconds: setting step_width to 1 and skip_leadin_fadeout to 0.
Analyzing 5 segments
# 5 / 1000 (ETA: 0:12:41): music/metal/metal.00096.au.wav
22050 Hz, 1 channel(s), 661794 samples
Duration < 45 seconds: setting step_width to 1 and skip_leadin_fadeout to 0.
Analyzing 5 segments
```

Figure 4.2: Screen shot of a log file output for running Support Vector Machine on Linux Ubuntu 16

performed on different development environments versions. We discussed that no significant changes happened when changing the version of the existing platform while reproducing machine learning algorithms.

In this chapter we concluded that operating systems don't have any influence on reproducing machine learning results. An additional experiments were made trying to reproduce the original study script of Lidy and Rauber [LR05] by running their Python 2.7 script in different operating systems. We came to a conclusion that the filing system for Linux, Windows and Mac operating systems is different, which provided different accuracy results while executing the same Python 2.7 script. After forced sorting of the audio files added in the code, all of the operating systems performed the same while executing Support Vector Machine, Naive Bayes and Random Forest algorithms.

In the next chapter a critical reflection of this thesis will be provided summarizing all the pros and cons that come when reproducing a machine learning results while changing the operating system or their development environment.Further more in the next chapter we suggest some future work steps that can be taken for better progress in reproducibility work dedicated in reproducing machine learning algorithms.

OS	OS Version	Environment	Environment Version	SVM	NB	RF
Windows	10	Python	2.7	65.60%	45.30%	52.90%
			3.6	65.60%	45.30%	52.90%
		Weka GUI	3.6	70.20%	50.00%	62.00%
			3.7	70.20%	50.00%	61.20%
			3.8	70.20%	50.00%	61.20%
		Weka API	3.6	71.80%	49.80%	62.40%
			3.7	71.80%	49.80%	62.10%
			3.8	71.80%	49.80%	62.10%
		RapidMiner	7.5	72.40%	48.90%	39.40%
		R	3.3.2	70.90%	52.00%	61.00%
	8	Python	2.7	65.60%	45.30%	52.90%
		Weka GUI	3.7	70.20%	50.00%	61.20%
		Weka API	3.7	71.80%	49.80%	62.10%
		RapidMiner	7.5	72.40%	48.90%	39.40%
		R	3.3.2	70.90%	52.00%	61.00%
	7	Python	2.7	65.60%	45.30%	52.90%
		Weka GUI	3.7	70.20%	50.00%	61.20%
		Weka API	3.7	71.80%	49.80%	62.10%
		RapidMiner	7.5	72.40%	48.90%	39.40%
		R	3.3.2	70.90%	52.00%	61.00%
Linux	Debian 6.8	Python	2.7	65.60%	45.30%	52.90%
		Weka GUI	3.7	70.20%	50.00%	61.20%
		Weka API	3.7	71.80%	49.80%	62.10%
		RapidMiner	7.5	72.40%	48.90%	39.40%
		R	3.3.2	70.90%	52.00%	61.00%
	Ubuntu 16	Python	2.7	65.60%	45.30%	52.90%
		Weka GUI	3.7	70.20%	50.00%	61.20%
		Weka API	3.7	71.80%	49.80%	62.10%
		RapidMiner	7.5	72.40%	48.90%	39.40%
		R	3.3.2	70.90%	52.00%	61.00%
Mac OS	Sierra	Python	2.7	65.60%	45.30%	52.90%
		Weka GUI	3.7	70.20%	50.00%	61.20%
		Weka API	3.7	71.80%	49.80%	62.10%
		RapidMiner	7.5	72.40%	48.90%	39.40%
		R	3.3.2	70.90%	52.00%	61.00%

Table 4.17: Summary results from both of the experimental schemes. We can see that by reproducing a machine learning results using a one time 10 fold cross validation in most of the cases the changes in the implementation does change the accuracy result. On the other hand by changing the operating system on which the implementation run doesn't influence the reproducibility of machine learning results in cases when the models which are reproduced are using a specific dataset input ("csv" file)

CHAPTER 5

Conclusion and future work

The aim of the reproducibility is to achieve the same prediction result as the main research. Scientists need to be aware of the fact that different operating systems and different development environments can alter the prediction accuracies of the model. On another note, by only changing the versions of the initial development environment or operating system, same prediction results are achieved.

In this thesis, we observed the influence of development environments such as R, RapidMiner, Python and Weka on reproducibility on machine learning results. Also we analyzed the changes that happen in machine learning results by running on different operating systems such as Windows, Linux and Mac OS and theirs different versions (Windows 10, Windows 8, Windows 7, Linux Debian 5.8 and Linux Ubuntu 16).

The suggestion concluded in this master thesis, is that in order to create reproducible machine learning results, the scientists need to be aware of the different folds which are created by different development environments. By the numerous experiments created, we proved that each environments use different order of the instances while creating the test and the training fold in cross validation, even though a random seed of 1 was given to each of them. This caused the accuracy of the models to be different while been implemented in different environments. By doing a Kolmogorov Smirnov two sample statistical test, we can conclude that there is a statistical significant difference between the differences in distribution for each of the folds created by the Python implementation and the Weka GUI/API implementation. All of the others pairs of tested implementations presented a p-value larger than 0.05 showing that there is not a statistical significant difference between the differences in distribution of the folds created.

With this being said, the researchers need to be very careful when trying to create a reproducible machine learning models using a cross validation implementation, having in mind that those study will most likely not give a reproducible results while been implemented in different development environment. Our recommendation would be to

provide a specific test and train dataset together with the reproducible code and a specific details about the development environment version and packages which were used in order to get a reproducible machine learning results.

Furthermore, when using a Weka environment for a research to be reproducible, the specific Weka implementation need to be provided, due to the fact that Weka GUI and WekaAPI in some cases calculated different results for the same experiment. The differences between the Weka API and Weka GUI are smaller then those across different development environments. Because of lack of documentation provided for the Weka libraries we can only assume that the differences between this two implementations lies either in the random generator or in different test/training folds created by the implementations. Even though both of the implementations give slightly different accuracy results, the difference in the distribution of the folds they create is not statistically significantly different.

By analyzing the experiments dedicated to find the influence that operating systems have on reproducing a machine learning algorithm we came to realization that changing the operating system and its version doesn't affect the machine learning model results while using a specific input data file. On the other hand, by trying to reproduce the study of Lidy and Rauber [LR05], we came to a realization that trying to reproduce a study which reads any type of files from an operating system influences the machine learning accuracy. This is caused by the fact that each operating system have a different file reading system which sorts the instances in a different order. Different versions of an operating system doesn't have any influence on the accuracy of a machine learning model.

Using the PRIMAD model, by changing the (P) Platform when reproducing machine learning algorithms, certain characteristics of the specific operating system need to be taken into consideration such as different filing system. While changing the (I) implementation of the development environment in reproducing a model, we need to be aware that the cross validation evaluation produces different training and testing folds in each environment, which effect the reproducibility of machine learning results.

Due to the fact that this thesis focused on reproducibility on a deeper level than the studies before, it opens a new door for future research in many aspects. Future work could focus on the attempt to gather and explore different type of machine learning results calculated from regression models and creating a similar study like this one with an aim for a brother understanding on the influence of environments and operating systems when reproducing machine learning results. We would expect that operating systems would not have an influence in reproducing regression results one hand, but it would be very interesting the analyze the influence of the different environments and their settings for regression models. Would the libraries for regression models for each implementation have any impact on the experiments prediction outcome? By going a similar research for reproducibility on regression models we will confirm all the hypothesis stated in this study.

Furthermore, different parts of the PRIMAD system can be explored while fixing the

research objective (R) and the method (M) and exploring the influence of the other variables such as Data(D), the Actor(A) and the Platform(P) on reproducibility of machine learning algorithms. Also, it would be very interesting to explore the particular case of the influence of different type of dataset inputs on a model.

Considering the importance of reproducibility in scientific community today, each study done in this area will influence the determination of and methods when creating reproducible studies based on machine learning algorithms.

CHAPTER 6

Appendix

6.1 Appendix A - Implementation in Weka GUI

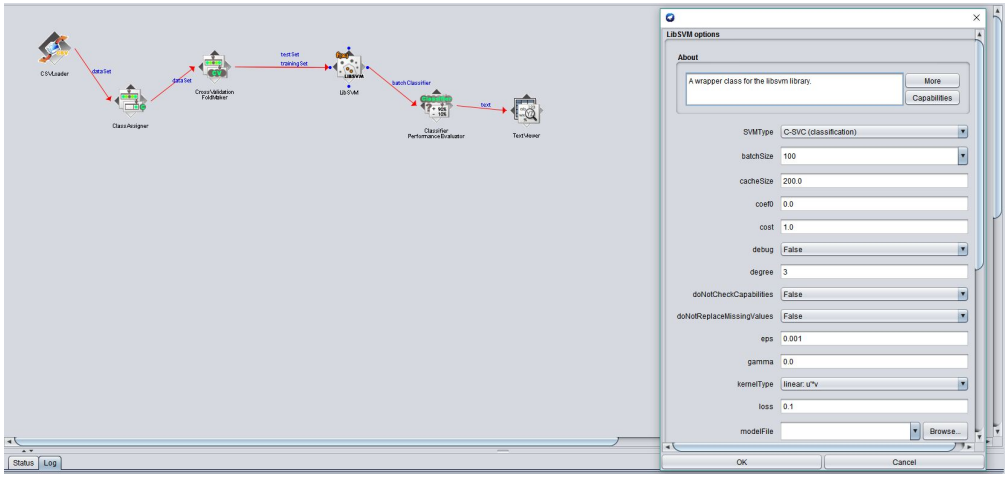

Figure 6.1: Support Vector Machine (LibSVM) used in Weka GUIusing Weka Knowledge Flow. We start at reading the csv file using the "CSVReader" operator. After importing the input file we define the class label in the file using "ClassAssigner" operator. A cross validation evaluation is implemented using the "CrossValidationFoldMaker" operator which splits the data 10 different folds and send them to the "LibSVN" operator which represent the Support Vector Machine model. A "classifierPerformanceEvaluatior" operator is used to calculate the performance measures of the model.

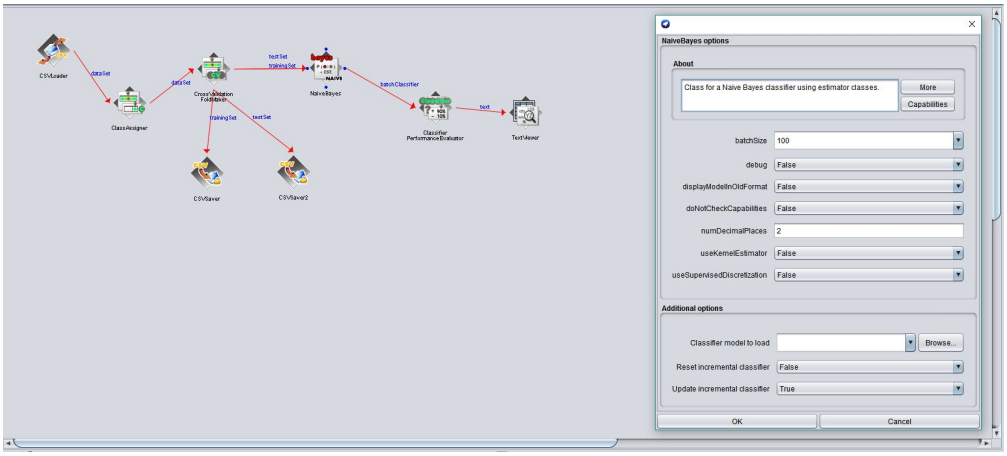

Figure 6.2: Naive Bayes algorithm used in Weka GUI using Weka Knowledge Flow. We start at reading the csv file using the "CSVReader" operator. After importing the input file we define the class label in the file using "ClassAssigner" operator. A cross validation evaluation is implemented using the "CrossValidationFoldMaker" operator which splits the data 10 different folds and send them to the "NaiveBayes" operator which represent the Support Vector Machine model. A "classifierPerformanceEvaluatior" operator is used to calculate the performance measures of the model.

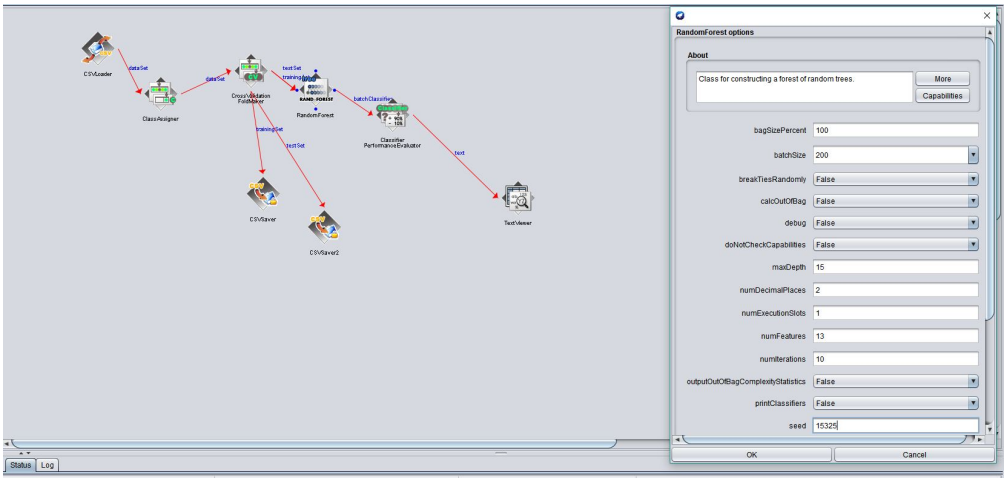

Figure 6.3: Random Forest algorithm used in Weka GUIusing Weka Knowledge Flow as presented int this Figure. We start at reading the csv file using the "CSVReader" operator. After importing the input file we define the class label in the file using "ClassAssigner" operator. A cross validation evaluation is implemented using the "CrossValidationFoldMaker" operator which splits the data 10 different folds and send them to the "RandomForest" operator which represent the Support Vector Machine model. A "classifierPerformanceEvaluatior" operator is used to calculate the performance measures of the model.

6.2 Appendix B - Implementation in Weka API

Weka API implementation is done using Java 8, Eclipse Mars IDE. Firstly we read the "arff" file as Instances. We set the parameters for each model separately and do a *crossValidatinModel* function to fit and predict the model and give results stored in *forPredictionPrintLog*.

```
package supportvectormachine;

import java.util.Random;

import weka.classifiers.Evaluation;
import weka.classifiers.bayes.NaiveBayes;
import weka.classifiers.functions.SMO;
import weka.classifiers.trees.RandomForest;
import weka.core.Instances;
import weka.core.converters.ConverterUtils.DataSource;

public class Classification {

public static void main(String args[]) throws Exception {
DataSource source = new DataSource(
"C:/Users/Martina/Desktop/algorithamsMT/
rp_extract-master/test.arff");
Instances dataset = source.getDataSet();
dataset.setClassIndex(dataset.numAttributes() - 1);
System.out.println(dataset.numAttributes());
if (dataset.classAttribute().isNominal()) {
dataset.stratify(10);

}

SMO supportVectorMachine = new SMO();

supportVectorMachine.setC(1.0);
supportVectorMachine.setRandomSeed(1);

supportVectorMachine.buildClassifier(dataset);

System.out.println(supportVectorMachine.getRandomSeed());
int folds = 10;

System.out.println("CVM classification ");
```

```
Evaluation SVM = new Evaluation(dataset);

StringBuffer forPredictionsPrinting = new StringBuffer();
Boolean outputDistribution = new Boolean(false);

SVM.crossValidateModel(supportVectorMachine, dataset, folds, new Random(1),
forPredictionsPrinting, new weka.core.Range("first"),
outputDistribution);

System.out.println(SVM.toSummaryString());
System.out.println("Support Vector Machine + Correct" +
SVM.pctCorrect());
System.out.println("Support Vector Machine + Incorect" +
SVM.pctIncorrect());
System.out.println("Support Vector Machine confusion
matrix:" + SVM.toMatrixString());

// Naiva Bayes Classifier
System.out.println("Naiva Bayes Classifier ");
StringBuffer forPredictionsNBPrinting = new StringBuffer();
weka.core.Range attsToOutput = null;
Boolean outputNBDistribution = new Boolean(false);

NaiveBayes naiveClassification = new NaiveBayes();
System.out.println("Naive batch size" +
naiveClassification.getCapabilities());
naiveClassification.buildClassifier(dataset);
Evaluation NB = new Evaluation(dataset);
NB.crossValidateModel(naiveClassification, dataset,
folds, new Random(1), forPredictionsNBPrinting,
new weka.core.Range("first"),
outputNBDistribution);

System.out.println(NB.toSummaryString());
System.out.println("NaiveBayes + Correct" + NB.pctCorrect());
System.out.println("NaiveBayes + Incorect" + NB.pctIncorrect());
System.out.println("NaiveBayes confusion matrix:" + NB.toMatrixString());

// Random Forest classifier
System.out.println("RandomForest Classification");
RandomForest forest = new RandomForest();

forest.setNumTrees(10);
```

```
forest.setSeed(15325);
forest.setMaxDepth(15);
forest.setNumIterations(10);
forest.setPrintClassifiers(true);
forest.setNumFeatures(13);

forest.buildClassifier(dataset);
Evaluation RND = new Evaluation(dataset);
StringBuffer forPredictionsRFPrinting = new StringBuffer();
Boolean outputRFDistribution = new Boolean(false);

RND.crossValidateModel(forest, dataset, folds,
new Random(1), forPredictionsRFPrinting, new weka.core.Range("first"),
outputRFDistribution);
System.out.println(forPredictionsRFPrinting);
System.out.println(RND.toSummaryString());
System.out.println("Random Forest + Correct" + RND.pctCorrect());
System.out.println("Random Forest + Incorect" + RND.pctIncorrect());
System.out.println("Random Forest confusion matrix:" + RND.toMatrixString());

}
}
```

6.3 Appendix C - Implementation in Python 2.7

6.3.1 Setting up the model and displaying the results of a cross validation in Python for one class labels

We are reading the model that needs to be run from the command line and fit and predict that particular model using 10 folds cross validation evaluation defined in a function cross-validate.

```
        if do_crossval:
print "CROSS-VALIDATION:"
if not args.train:

if (args.SVM):
model = svm.SVC(kernel='linear',C=1,class_weight= {0:1, 1:1,2:1,3:1,4:1,5:1,6:1,

if(args.GAUS):
model = GaussianNB()
if(args.RAND):
model=RandomForestClassifier(n_estimators=10,
```

```
random_state=15325,max_depth=15)
print "Model is", model
if not args.multiclassfile:

import pandas as pd

my_df = pd.DataFrame(features)
my_df["class"]=classes_num
my_df.to_csv('my_csv.csv', index=False, header=False)

dataset = pd.read_csv('weka_data/fullDataForPythonNoHeader.csv',
 header=None, sep=',')

y = np.array(dataset[[168]].values)

y1 = y.flatten()
x = dataset.iloc[:, :168].as_matrix()

acc = cross_validate(model, features, classes_num, 10, crossval_measure)
print "Fold " + crossval_measure + ":", acc
print "Avg " + crossval_measure + " (%d folds): %2.2f %% (std.dev.: %2.2f)"
```

6.3.2 Cross Validation implementation in Python for one class labels

Creating a function cross-validate which gets the parameters such as the model, the features of the dataset the labeled classes, the number of folds that need to be created and the measure type. In this function we print the folds as they are generated, because of the need to analyze their variance and statistical significant difference. Furthermore, we predict the model and print the accuracy, kappa statistic, Mean squared error and Mean absolute error as a results.

```
def cross_validate(model, features, classes, folds=10,
 measure='precision'):

from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score
from sklearn.metrics import cohen_kappa_score
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import explained_variance_score
skf = StratifiedKFold(n_splits=10,random_state="1")
#print  "stratified folds:" + skf
```

```
skf.get_n_splits(features,classes)
VariancesTrainingSetFolds = []
VariancesTestingSetFolds = []
VarianceScore =[]
for train_index, test_index in skf.split(features, classes):

print "TRAIN:", train_index, "TEST:", test_index
VariancesTrainingSetFolds.append(np.var(features[train_index]))
VariancesTestingSetFolds.append(np.var(features[test_index]))
print "Variance List of training samples for each ford:"
 ,VariancesTrainingSetFolds
print "Variance of test samples for each fold",
VariancesTestingSetFolds

predicted = cross_val_predict(model,features,classes,cv=skf)
print "Predicted score from cross validation :",predicted,
"real values:",classes
np.savetxt("PredictedValuesSVM2.csv",predicted,
 delimiter=',', newline='\n', header='', footer='', comments='#')
cnf_matrix = confusion_matrix(classes, predicted)
print("matrix:", cnf_matrix)
kappa =  cohen_kappa_score(predicted, classes)
print "Kappa statistics:",kappa
print "Mean absolute error ", mean_absolute_error(classes,predicted)

print "Mean sqared error " , mean_squared_error(classes,predicted)



from sklearn.model_selection import cross_val_score

return cross_val_score(model, features, classes, scoring=measure, cv=skf)
```

6.4 Appendix D - Implementation in Rapid Miner

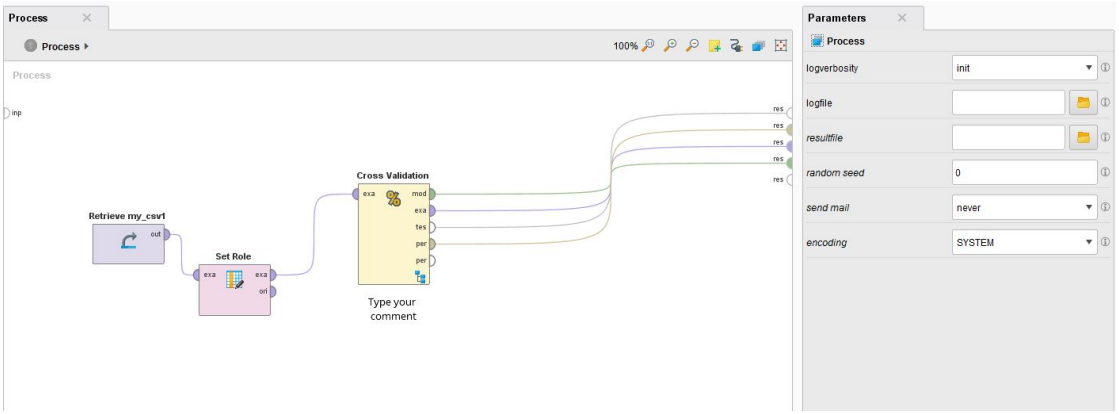

Figure 6.4: For our experiments we use Rapid Miner Studio 7.3, using RapidMiner implemented operators for each of the machine learning models. Each of the processes is retrieving the data set from a "csv" file, from where a SetRole operator is implemented. A Set Role operator is setting the class feature as a target label for the model. Connected to this operator is the operator which does a 10-fold cross validation operator using a Stratified Sampling with a random seed of 1. Every experiment implemented in RapidMiner has the same base.

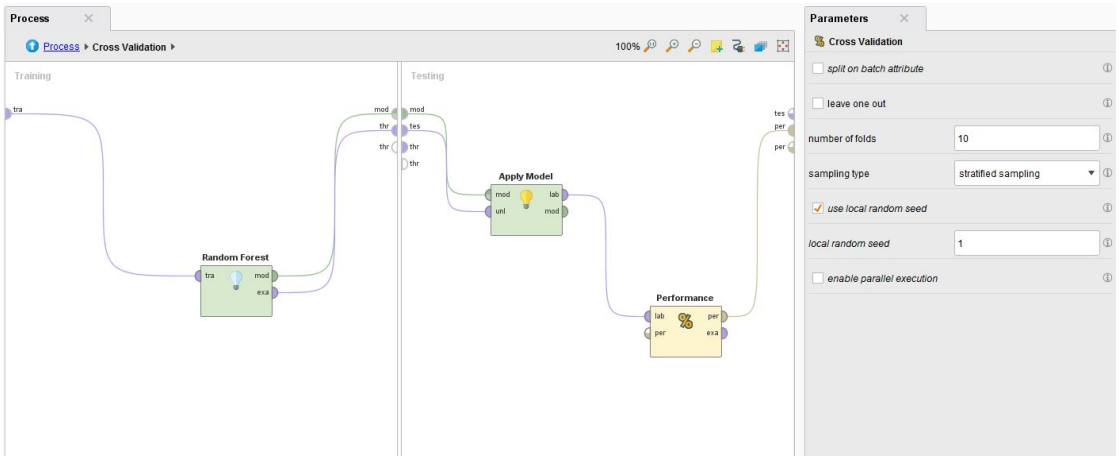

Figure 6.5: Inside of the Cross Validation operator, a training side and a testing side are presented. The training side includes an operator representing a specific model that we want to train and the testing side includes the Apply Model and Performance operators witch give us the performance of the model for every fold. For implementation of Support Vector Machine and Naive Bayes algorithms only the operator in the training side was changed.

6.5 Appendix E - Implementation in R

In the R implementation we set the seed of 1 at the beginning and fit each model using different R library as (libsvm,caret and Random Forest). For each of the models we extract the accuracy, the kappa value and the Mean squared error.

```
require(caret)
library(caret)

library(libsvm)
library(e1071)
library(randomForest)
musicFeatureDatabase <- read.csv("C:/Users/Martina/Desktop/algorithamsMT/
rp_extract-master/my_csv1.csv")

#Support Vector Machine model
set.seed(1)
svm.fit = svm(class~ .,data=musicFeatureDatabase,kernel="linear",
cost=1,scale=FALSE,cross=10,type="C-classification")
train_control <- trainControl(method="cv", number=10,
savePredictions = TRUE)
fitSVN <- train(musicFeatureDatabase[,-169],musicFeatureDatabase[,169],
method = "svmRadial",trControl=train_control,C=1.0)
predictions <- fitNB$pred

listAcurracies <- svm.fit$accuracies
SVMAccuracy <- mean(listAcurracies)

#Naive Bayes model
set.seed(1)
train_control <- trainControl(method="cv",
number=10,savePredictions = TRUE)

fitNB <- train(musicFeatureDatabase[,-169],musicFeatureDatabase[,169],
method = "nb",trControl=train_control)
AccuracyNB<-fitNB$results[2,4]
KappaValueNB <- AccuracyNB<-fitNB$results[2,5]

# Random Forest Model

fit.RandomForest <- train (
 musicFeatureDatabase[,-169],musicFeatureDatabase[,169]
,method = 'cforest',trControl=train_control,rand=1)
predictions <- fit.RandomForest$pred
```

List of Figures

List of Tables

Bibliography

[BGV92] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

[BJ96] Frederick P Brooks Jr. The computer scientist as toolsmith ii. *Communications of the ACM*, 39(3):61–68, 1996.

[Bre01] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[CKK+15] John T Cacioppo, Robert M Kaplan, Jon A Krosnick, James L Olds, and Heather Dean. Social, behavioral, and economic sciences perspectives on robust and reliable science. 2015.

[CL11] Chih-Chung Chang and Chih-Jen Lin. Libsvm: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[CP16] Christian Collberg and Todd A Proebsting. Repeatability in computer systems research. *Communications of the ACM*, 59(3):62–69, 2016.

[Dav15] Meyer David. Support vector machines. *The Interface to libsvm in package e1071*, 2015.

[FFG+60] Sir Ronald Aylmer Fisher, Ronald Aylmer Fisher, Statistiker Genetiker, Ronald Aylmer Fisher, Statistician Genetician, Great Britain, Ronald Aylmer Fisher, and Statisticien Généticien. The design of experiments. 1960.

[FFR16] Juliana Freire, Norbert Fuhr, and Andreas Rauber. Reproducibility of data-oriented experiments in e-science (dagstuhl seminar 16041). In *Dagstuhl Reports*, volume 6. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[GFI16] Steven N Goodman, Daniele Fanelli, and John PA Ioannidis. What does research reproducibility mean? *Science translational medicine*, 8(341):341ps12–341ps12, 2016.

[GHJ+12] Ed HBM Gronenschild, Petra Habets, Heidi IL Jacobs, Ron Mengelers, Nico Rozendaal, Jim Van Os, and Machteld Marcelis. The effects of freesurfer version, workstation type, and macintosh operating system version on anatomical volume and cortical thickness measurements. *PloS one*, 7(6):e38234, 2012.

[Har95] Juris Hartmanis. On computational complexity and the nature of computer science. *ACM Computing Surveys (CSUR)*, 27(1):7–16, 1995.

[Hor12] Kurt Hornik. The comprehensive r archive network. *Wiley Interdisciplinary Reviews: Computational Statistics*, 4(4):394–398, 2012.

[Kov07] Jelena Kovacevic. How to encourage and publish reproducible research. In *In poceedings of the 2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*, volume 4, pages IV–1273. IEEE, 2007.

[LR05] Thomas Lidy and Andreas Rauber. Evaluation of feature extractors and psycho-acoustic transformations for music genre classification. In *Proceedings of the Sixth International Conference on Music Information Retrieval (ISMIR 2005)*, pages 34–41, London, UK, September 11-15 2005.

[NGP09] Eilen Nordlie, Marc-Oliver Gewaltig, and Hans Ekkehard Plesser. Towards reproducible descriptions of neuronal network models. *PLoS Comput Biol*, 5(8):e1000456, 2009.

[PE09] Roger D Peng and Sandrah P Eckel. Distributed reproducible research using cached computations. *Computing in Science & Engineering*, 11(1):28–34, 2009.

[Pen11] Roger D Peng. Reproducible research in computational science. *Science*, 334(6060):1226–1227, 2011.

[Pop05] Karl Popper. *The logic of scientific discovery*. Routledge, 2005.

[Rou14] David De Roure. The future of scolary comunications.insights. page 233–238, 2014.

[SNTH13] Geir Kjetil Sandve, Anton Nekrutenko, James Taylor, and Eivind Hovig. Ten simple rules for reproducible computational research. *PLoS Comput Biol*, 9(10):e1003285, 2013.

[SS11] Steven Shapin and Simon Schaffer. *Leviathan and the Air-Pump: Hobbes, Boyle, and the Experimental Life (New in Paper)*. Princeton University Press, 2011.

[TC02] George Tzanetakis and Perry Cook. Musical genre classification of audio signals. *IEEE Transactions on speech and audio processing*, 10(5):293–302, 2002.